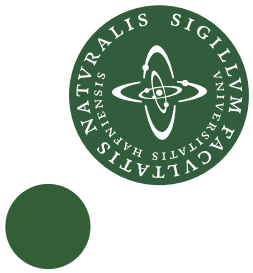
FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN



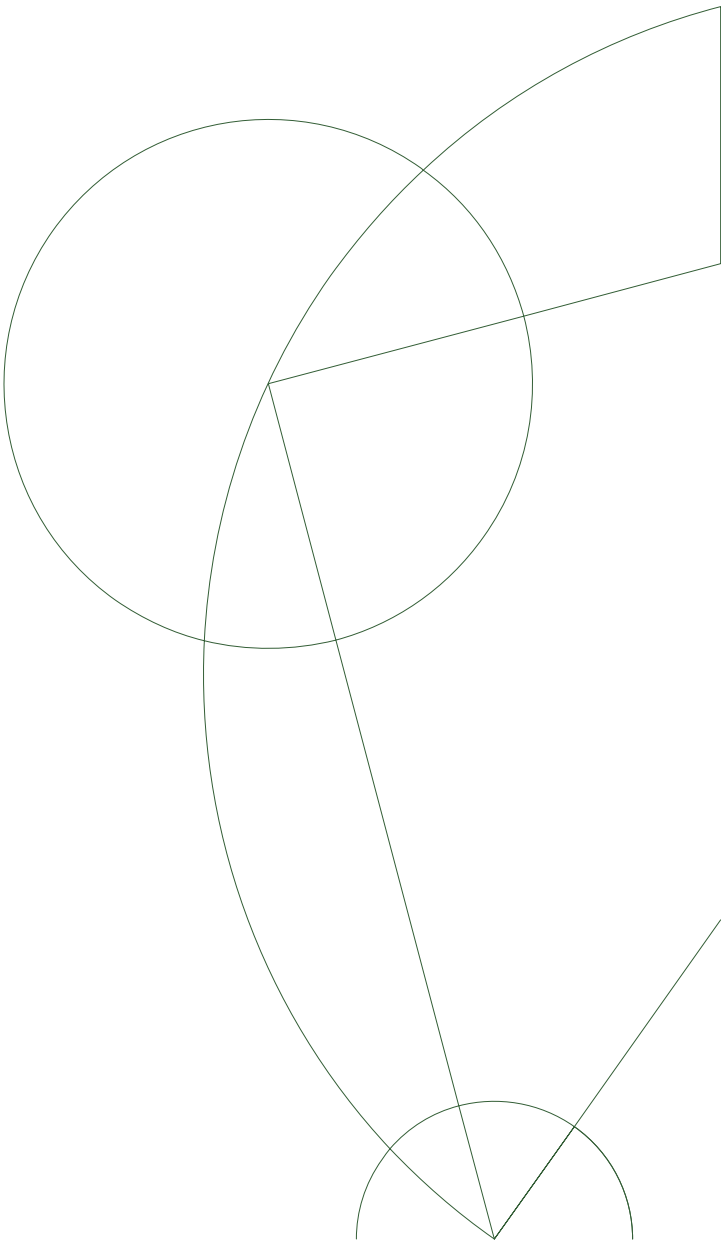
Master Thesis

Financial Forecasting with Long Short-Term Memory Networks

March 5th, 2018

Jens Grud <rvq134@alumni.ku.dk>

Supervisor
Christian Igel <igel@di.ku.dk>



Abstract

Deep neural networks have improved the state-of-the-art across a wide variety of domains. This thesis revisits the application of recurrent neural network (RNN) architectures to financial time series. Four RNN architectures implemented in the TensorFlow machine learning framework are considered, namely basic RNNs, Long Short-Term Memory (LSTM) networks with and without peephole connections, as well as networks with Gated Recurrent Units (GRUs). The four RNN architectures are benchmarked against recently published RNN results and stochastic models applied in the financial industry on three different market indices over a 6 year period.

In our experimental comparison, the basic LSTM architecture performed on par with its peephole counterpart and the GRU architecture compared at a 0.05 significance level. The LSTMs and the GRU performed significantly better than the basic RNN. The GRU architecture was most stable in terms of lower standard deviations and faster in terms of actual compute time across all runs. It performed as well as recently published results achieved with LSTMs combined with an elaborate preprocessing. Therefore, the GRU architecture is proposed as the RNN architecture of choice for the considered type of application. However, the stochastic models of the financial industry and the naive benchmark measure of predicting the value at time $t-1$ at time t (`t-1`) still outperformed all RNN architectures.

Different ways to improve the RNN approach are explored. Experimental results are presented 1) comparing the Resilient Backpropagation (Rprop) and Improved Resilient Backpropagation (iRprop) optimization algorithms to the tuned Adam algorithm, 2) performing predictions with a rolling training phase, and 3) adding more training samples. The Rprop algorithms with default parameters performed as well as Adam with tuned hyperparameters. Overall, the iRprop$^+$ method gave the best results. Neither adding more training samples nor performing predictions with a rolling training phase improved the performance.

Contents

1 Introduction

Recent advances in deep neural networks have shown them to be very versatile in performing accurate predictions across a wide variety of domains [33]. Recurrent neural networks (RNN) [17] with Long Short-Term Memory (LSTM) [21, 19] in particular have been widely adopted for modelling complex, sequential real world data such as language and time series [20, 3], where the meaning at a given step is highly dependent on its context.

Combining these developments with the prospects of successful financial forecasting makes for a promising endeavor. However the volume of publicly reported applications of recurrent networks within financial forecasting is limited and the results that do exist lack a common framework for reporting error measures, making it difficult to evaluate relative performance.

Therefore, using recurrent neural networks with Long Short-Term Memory, we wish to build a state-of-the-art model for performing one-step-ahead stock market closing price prediction. In the process, we 1) outline the motivation for the LSTM architecture and apply established optimization techniques, 2) benchmark predictive accuracy against other recurrent architectures while reasoning about results and 3) document approach, network topology, configuration and results to an extend to which all can be reproduced and serve as the starting point or baseline for similar attempts in the domain.

In total, four recurrent architectures are trained and compared on 3 different data sets over a 6 year period. The results are benchmarked against recently published LSTM results and stochastic models of the financial sector toolbox.

The contribution of this thesis is fourfold:

- First, to summarize the current state of recurrent neural networks applied to financial time series by reporting results and grouping them by data set, time frame and error measure. This also serves as the foundation on which we will build our own experiment setup.
- Second, to build and optimize a state-of-the-art model for performing one-step-ahead stock market closing price prediction with recurrent neural networks while documenting the approach to a degree where the prediction pipeline, network topology and results can be reproduced. This could possibly serve as the baseline for further explorations in the field.
- Third, to benchmark results from the RNN models against similar approaches in the existing literature and the stochastic models in the real world financial sector, represented by RiskButler.

- Finally, to present a discussion section with suggestions for further work and concrete results on 1) a comparison between the Resilient and Improved Resilient Backpropagation optimization algorithms [23, 24] and the tuned Adam optimization algorithm, 2) approximating real life conditions, by assessing how performing a single training pass compares to a training pass with a rolling time horizon, 3) adding more training samples, and lastly 4) expanding the network architecture by adding additional feed forward layers both before and after the recurrent layers.

The remainder of the thesis is structured as follows: Section 2 outlines the current state of machine learning models applied to financial data in general and RNNs in particular. Section 3 briefly describes some important financial concepts and theory needed for our experiment before Section 4 presents the theoretical foundation and motivation for applying recurrent neural networks to time series prediction.

Section 5 describes the experiment setup in detail and Section 6 presents the results and a comparison with benchmark models. Finally a discussion of results including suggestions for further work is presented in Section 7 before conclusions are drawn in Section 8.

2 Current state-of-the-art

This section briefly outlines the current state-of-the-art for financial time series modelling using machine learning algorithms in general and recurrent neural networks with Long Short-Term Memory (LSTM) in particular. Without a common framework it is difficult to evaluate relative performance in a consistent way, and hence particular attention is payed to data set, input variables, forecasting time frame and error measures as this will serve as the basis on which we conduct our own experiment.

It is beyond the scope of this thesis to systematically and exhaustively review and categorize all existing literature. Instead, the purpose of this section lies in enclosing method for determining current state in detail.

2.1 Machine learning models for financial forecasting

A recent survey [32] conducts a review on the existing literature in the domain of interest. The reviewed papers are grouped according to algorithm employed, forecasting time frame, input variables and evaluation technique. The conclusions drawn are the following:

- "The majority of publications tries to make one day ahead predictions using stock index data."

- "Lagged index data and technical indicators have been identified as the most popular input parameters"
- "Artificial Neural Networks (ANNs) have been identified as the dominant machine learning technique in this area."

Using this as foundation, we perform a search with similar desired outcome structure of Long Short-Term Memory networks applied to financial data.

2.2 Long Short-Term Memory networks for financial forecasting

In general, neural networks have been applied within finance for a long time [12]. In particular, recurrent neural networks applied to financial data is a relatively new and unexplored field and determining current state-of-the-art proved difficult, as data sets and error measures vary, details on significance are omitted and overall very little documentation on methodology and implementation is provided[1].

The approach taken in this thesis relies on a semi structured keyword search, based on manual filtering conducted on various search engines. For each source found, the same keyword search was applied to references deemed relevant until a circular dependency seemed to be reached. The procedure is illustrated in Algorithm 1. Again, the purpose was to gain a broader picture of the landscape, not to perform a systemic and exhaustive review. Table 6 in Appendix A presents all results grouped by data set, input variables, forecast horizon and error measures. Additionally, a column is devoted to citations count.

[1] Adding to this, a basic Google search will produce ample results of misleading tutorials and in some cases plain wrong explanations that one should only trust and utilize with a good measure of caution.

Data: Search engine queries with combinations of keywords/phrases: *"lstm"*, *"rnn"*, *"financial time series"*, *"financial forecasting"*, *"stock prediction"*

Result: Filter out everything not RNN related, not applied to financial data/time series, not containing reported results or with no citations.

while *Not reached cyclic references between papers* **do**
  read paper;
  if *contains relevant reference by filtering criterion* **then**
    apply algorithm to reference;
  else
    continue next paper;
  end
end

Algorithm 1: Algorithm applied for determining current state-of-the-art using RNN networks for financial forecasting.

The search confirms what [32] established; most papers perform predictions with a one day horizon and use lagged index data sometimes in combination with technical indicators as input variables. However, error measures reported in most cases provide little basis for benchmarking as they are either non-existent or feature scaling dependent. In general the citations count is small suggesting that the domain is relatively unexplored.

The best documented prediction approached and reported error measures suited for comparison is found in [5]. This will serve as the basis for our experiment setup in Section 5.

3 Financial time series forecasting

At a logical level, financial forecasting has to be a difficult problem to solve as it involves one of the most basic human motivators: direct and significant financial rewards. At a research backed level, the generally accepted semi-strong Efficient Market Hypothesis [37] supports this intuition by stating that prices already reflect all information available and thus cannot be anticipated; the amount of available data is overwhelming, the signal-to-noise ratio is low and the conflicting interests of speculators only looking for their next trading opportunity makes it close to impossible to identify repeating patterns.

However, professional traders already rely highly on the technical analysis methodology in aiding their trading strategy [45] and entire companies truly are build around forecasting movements in the financial markets[2] – all

[2]Risk Butler [42] is one such company that by estimating parameters of stochastic differential equations seeks to forecast stock market movements.

suggesting that these patterns do indeed exist.

3.1 Tools of the trade

In practice, two stock prediction methodologies prevail when it comes to understanding and predicting future movements of financial markets.

3.1.1 Fundamental analysis

In the fundamental analysis [25], the investor tries to asses the intrinsic value of a given stock by understanding the company, product(s) and industry in relation to various quantifiable financial key metrics and overall macro and micro economic trends.

3.1.2 Technical analysis

The technical analysis [26] focuses exclusively on statistical analysis of available market information, such as trading price and volume. This framework thus serves as a good foundation for building potential feature vectors for a model trying to forecast market movements.

Following the prediction methodology of [5], 8 commonly used technical indicators have been identified and chosen. For all definitions, n is the number of time steps, t is the time interval, C is the price at the end (closing) of the period, O the price at the beginning (opening), L the lowest price, H the highest and V the trading volume.

3.1.2.1 Simple Moving Average (MA) As one of the most used indicators, the Simple Moving Average is used to identify trends and generate potential buying and selling signals by the following definition:

$$\texttt{MA} = \sum_{t=1}^{n} C_t \tag{3.1.1}$$

3.1.2.2 Exponential Moving Average (EMA) The Exponential Moving Average tries to identify trends by smoothing the curve using the following regularization scheme:

$$\texttt{EMA}_t^n = \alpha C_t + (1-\alpha)\texttt{EMA}_{t-1} \tag{3.1.2}$$

$$\alpha = \frac{2}{1+n} \tag{3.1.3}$$

3.1.2.3 Rate of Change (ROC) The Rate of Change indicator will measure the percent change in price from one period to the next by the following definition:

$$\texttt{ROC}_t = \frac{O_t - O_{t-n}}{O_{t-n}} \tag{3.1.4}$$

3.1.2.4 Bollinger Bands (BBANDS) Bollinger Bands consist of a middle band defined as an exponential moving average and two outer bands usually set $k = 2$ standard deviations above and below:

$$\texttt{UBOL}_t^n = \texttt{EMA}_t^n + k\sigma^n \tag{3.1.5}$$

$$\texttt{LBOL}_t^n = \texttt{EMA}_t^n - k\sigma^n \tag{3.1.6}$$

3.1.2.5 Commodity Channel Index (CCI) The Commodity Channel Index provides a measure on a given stocks variation by the following definition:

$$\texttt{TP} = \frac{H + L + C}{3} \tag{3.1.7}$$

$$\texttt{CCI} = \frac{1}{0.015}\frac{\texttt{TP} - \texttt{MA(TP)}}{\sigma(\texttt{TP})} \tag{3.1.8}$$

where σ is defined as the mean absolute deviation around a central point, X:

$$\frac{1}{n}\sum_{i=1}^{n}|x_i - X| \tag{3.1.9}$$

3.1.2.6 Average True Range (ATR) As opposed to the indicators above, given high, low and closing price, the Average True Range provides an indication of the price volatility instead of the price trend by the following equation:

$$\texttt{TR} = \max\left\{(H_t - L_t), abs(H_t - C_{t-1}), abs(L_t - C_{t-1})\right\} \tag{3.1.10}$$

$$\texttt{ATR} = \frac{1}{n}\sum_{t=1}^{n}\texttt{TR}_t \tag{3.1.11}$$

$$\texttt{ATR}^t = \frac{\texttt{ATR}^{t-1}(n-1) + \texttt{TR}_t}{n} \tag{3.1.12}$$

3.1.2.7 Williams' %R (WILLR) The Williams' %R is used to establish entry and exit points in the market by comparing the closing price of a given stock to its high/low range over a time period n, where typically $n = 14$:

$$\texttt{WILLR} = \frac{H^n - C_t}{H^n - L^n}(-100) \tag{3.1.13}$$

Overall, the financial indicators thus help determine where the trend of the price and volatility is headed for a given stock on a time period longer than a day.

3.1.3 Stochastic processes

The analysis at RiskButler is build around stochastic differential equations for interpreting market data and forecasting market movements. A rigorous mathematical definition is beyond the scope of this thesis but the benchmark models and their definitions are provided below with a short explanation. The reader is referred to [22] for an in depth understanding of the Brownian motion and the drift and volatility coefficients referenced, and [8] for an empirical comparison of alternative stochastic models applied to finance.

3.1.3.1 Geometric Brownian Motion A stochastic process S_t is said to follow a Geometric Brownian Motion (GBM) if it satisfies the stochastic differential equation:

$$dS_t = \mu S_t dt + \sigma S_t dW_t \tag{3.1.14}$$

where W_t is a standard Brownian motion process[3] and μ and σ are constants known as the drift and volatility coefficients respectively.

3.1.3.2 CKLS 1992 A stochastic process S_t is said to follow a CKLS motion if it satisfies the stochastic differential equation:

$$dS_t = \kappa(\mu - S_t)dt + \sigma S_t^{\gamma} dW_t \tag{3.1.15}$$

where again, W_t is a standard Brownian motion process and now μ and κ are the drift coefficients and σ and γ volatility coefficients.

The models will be referenced throughout the thesis by their abbreviation, in particular in the results Section 6.

[3]The standard Brownian motion process is also known as a Wiener process, hence the W

4 Recurrent Neural Networks

This section presents the motivation for the recurrent network architecture with some common applications. The learning problem is defined, the equations for the forward pass of the basic recurrent network are given and the potential pitfalls explained and used to motivate the Long Short-Term Memory (LSTM) architecture.

Before outlining the equations of the forward pass of the LSTM, a brief summary of definitions used in existing literature is given and definitions used throughout the rest of the thesis established. Finally a subsection on peephole connections in the LSTM architecture and Gated Recurrent Units (GRUs) is presented.

4.1 Motivation

To motivate the properties of the recurrent network architecture, we start by introducing three concepts: cycles, unfolding and parameter sharing.

4.1.0.1 Cycles We define a cycle as the influence of a present value of a variable on its value at a future time step. Equation 4.1.1 gives an example of a cyclic dependency, where h is the state and f is a function that maps the state at time t to the state at time $t+1$ with the parameter θ. As can be seen, the recurrence appears because h at time t is dependent on h_{t-1}.

$$h_t = f(h_{t-1}, \theta) \tag{4.1.1}$$

4.1.0.2 Unfolding From this, we define unfolding as repeatedly applying the definition given in Equation 4.1.1 until we reach an expression that does not include dependencies on it self at a previous time step. An expression we can now represent by a traditional directed a-cyclic graph.

4.1.0.3 Parameter sharing Unfolding cycles allows us to illustrate the last concept; parameter sharing. Given a finite number of time steps $T = 3$, Equation 4.1.1 is unfolded into:

$$\begin{aligned} h_3 &= f(h_2, \theta) \\ &= f(f(f(h_0, \theta), \theta), \theta) \end{aligned} \tag{4.1.2}$$

As can be seen, the parameter θ remains unchanged at each unrolled cycle and is thus shared across each time step. This allows us to share statistical strength across different positions in time – or more intuitively, to recall reasoning about previous events when processing later ones. A

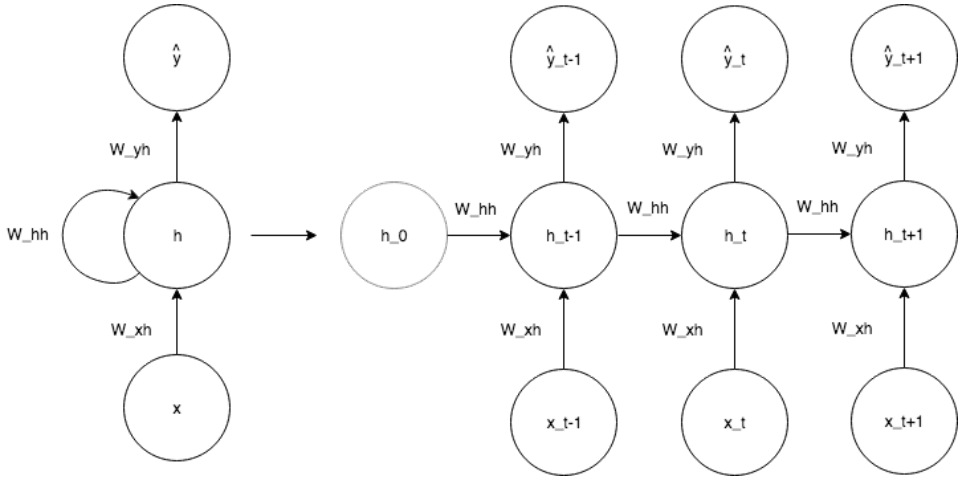


Figure 1: Unrolled basic recurrent network. Cycles are introduced between recurrent edges that connect adjacent time steps to allow for the current value of a variable to influence its future value. Activation function σ is applied between each time step.

property that is not available in a traditional feed forward network, where no dependencies between input vectors has to be assumed. This makes the recurrent architecture well suited for processing sequential data like time series and human language, where the meaning at a given time step depends highly on its context.

4.2 Basic recurrent networks

The basic recurrent network architecture leverages the properties of cycles and parameter sharing introduced above by introducing a hidden state. Figure 1 illustrates the unfolding of such a recurrent network, where:

- x_t is the input vector at time t
- h_t is the hidden state vector maintained by the network at a given time step t
- W_{hh} is the transition weight matrix of the hidden state
- W_{xh} and W_{yh} are the weight matrices of the input layer and output layer respectively
- σ is the (output) activation function.
- $\hat{y}_t$ is the output at time t.

Notice how weights W_{hh}, W_{xh}, W_{yh} are shared across time steps. The initial hidden state h_0 is typically initialized to zero. The full equations for the forward pass of the basic recurrent network are given below:

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t) \tag{4.2.1}$$

$$\hat{y}_t = W_{hy}h_t \tag{4.2.2}$$

4.2.1 The learning problem

We define our learning problem stated in Section 1 with the following: considering time series sampled at discrete time steps and a data set $\mathcal{S}$ with n such time series, we want to map each input sequence at time t to a corresponding output sequence at time $t+1$ of same length.

Equation 4.2.3 illustrates the problem:

$$\begin{aligned}\mathcal{S} = \Big\{ & \{x_1^1, x_2^1, \ldots, x_{T_1}^1\}, \{y_1^1, y_2^1, \ldots, y_{T_1}^1\}, \\ & \{x_1^2, x_2^2, \ldots, x_{T_2}^2\}, \{y_1^2, y_2^2, \ldots, y_{T_2}^2\}, \\ & \vdots \\ & \{x_1^n, x_2^n, \ldots, x_{T_n}^n\}, \{y_1^n, y_2^n, \ldots, y_{T_n}^n\} \Big\}\end{aligned} \tag{4.2.3}$$

where each series consists of d-dimensional input vectors $x_1^1, \ldots, x_T^i \in \mathbb{R}^d$ and corresponding output values $y_1^1, \ldots, y_T^i \in \mathbb{R}$. To assess predictive performance, we define the empirical error:

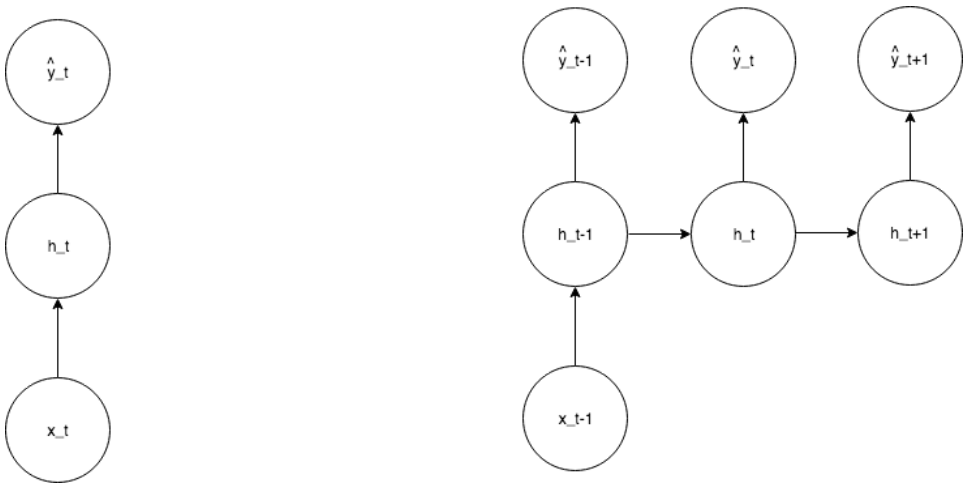
$$\sum_{i=1}^{n} \sum_{t=1}^{T_i} \mathcal{L}(y_t^i, \hat{y}_t^i) \tag{4.2.4}$$

where $\mathcal{L}$ denotes a point-wise loss function, that will measure the difference between expected and actual output y and $\hat{y}$.

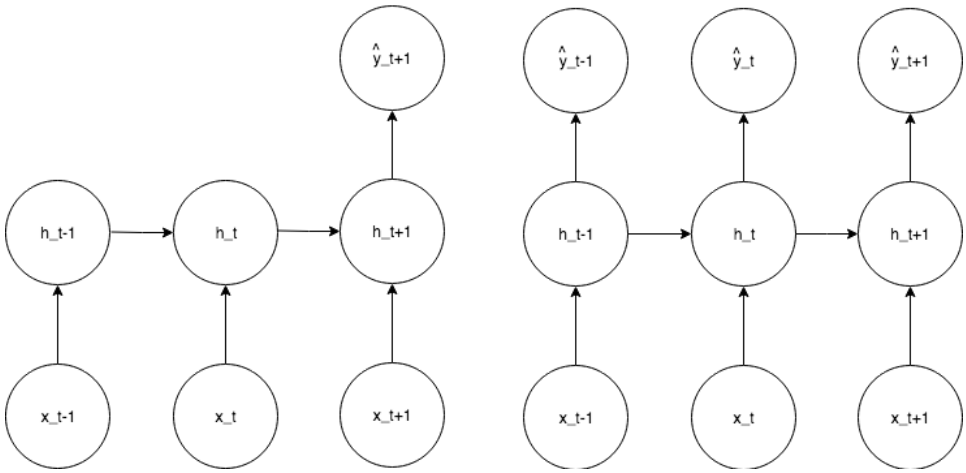
A wide variety of design patterns for recurrent neural networks exists and Figure 2 outlines the most common. In particular, the sequence classification architecture (many-to-one) is interesting, where looking only at the last output of the network, a mapping is done to a single relevant class.

4.2.2 Vanishing gradients

In practice, training basic recurrent networks as outlined in Equations 4.2.1 and 4.2.2 is known to be difficult, in particular due the vanishing gradient problem [39].



(a) One-to-one (left) and one-to-many (right)



(b) Many-to-one (left) and many-to-many (right)

Figure 2: Common design patterns for the recurrent neural network.

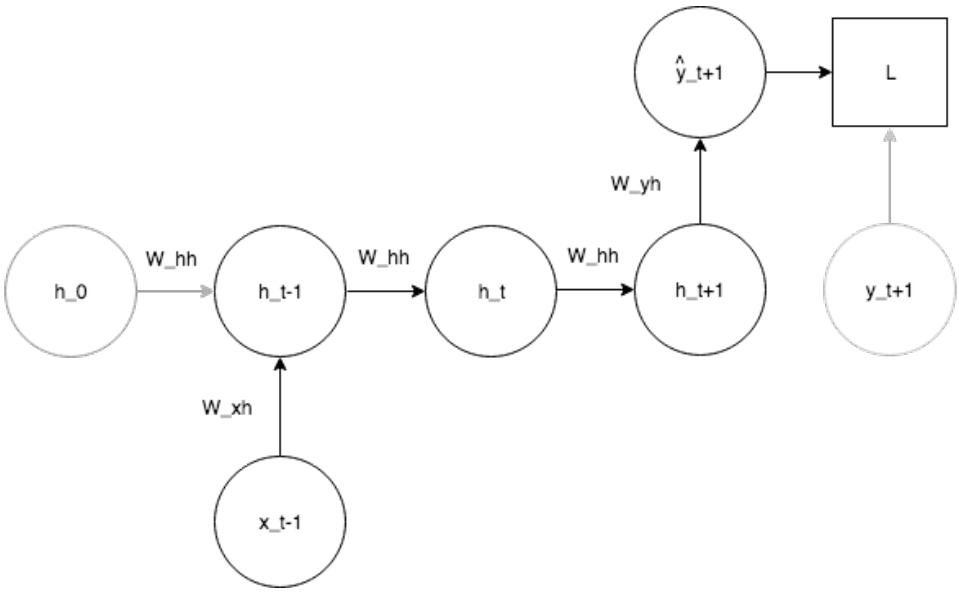


Figure 3: Basic recurrent architecture with input size $I = 1$, output size $K = 1$ and sequence length $T = 3$

To illustrate this, again, we define a point-wise loss function $\mathcal{L}(y, \hat{y})$, that will measure the difference between expected and actual output y and $\hat{y}$. Our objective is now to find the values for the parameters W_{hh}, W_{xh}, W_{yh} of the recurrent network that minimizes the loss function $\mathcal{L}$.

For simplicity, we will limit our discussion to the weight parameter W_{hh} and show why this is sufficient and we will assume a network with input size $I = 1$, output size $K = 1$ and a sequence of length $T = 3$, as illustrated in Figure 3.

4.2.2.1 The W_{hh} parameter By applying the chain rule of differentiation to Equations 4.2.1 and 4.2.2 and the loss function $\mathcal{L}$ with respect to W_{hh} at a single time step t we have:

$$\frac{\partial \mathcal{L}_t}{\partial W_{hh}} = \frac{\partial \mathcal{L}_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W_{hh}} \tag{4.2.5}$$

However, to compute the last term $\frac{\partial h_t}{\partial W_{hh}}$ we have to account for the implicit dependency of h_t on W_{hh} through h_{t-1} and h_{t-1} on W_{hh} through h_{t-2} where we reach initial hidden state vector h_0:

$$\frac{\partial \mathcal{L}_t}{\partial W_{hh}} = \frac{\partial \mathcal{L}_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W_{hh}} \tag{4.2.6}$$

Equation 4.2.7 recursively unrolls the value of h_t:

$$\begin{aligned} h_t &= f(W_{hh}h_{t-1}) \\ h_{t-1} &= f(W_{hh}h_{t-2}) \\ h_{t-2} &= f(W_{hh}h_0) \end{aligned} \tag{4.2.7}$$

which we compound to:

$$h_t = f(W_{hh}f(W_{hh}h_{t-2}))) \tag{4.2.8}$$

Applying the chain rule, the partial derivative of h_t with respect to h_{t-2} is then given by:

$$\frac{\partial h_t}{\partial h_{t-2}} = f'(W_{hh}f(W_{hh}h_{t-2})) \cdot W_{hh} \cdot f'(W_{hh}h_{t-2})) \cdot W_{hh} \tag{4.2.9}$$

which we can substitute into Equation 4.2.6.

Equation 4.2.9 thus separates the value of the gradient from the weight parameters and allows us to reason about the effect of different choices of activation functions and weight initialization strategies as we increase the number of time steps T.

Figure 4 plots the derivatives of the $tanh$ and σ activation functions and Equation 4.2.10 visualizes the magnitude of the components of Equation 4.2.9. As can be seen:

- σ' is bounded from above by 1/4, approximating zero for $|x| > 6$
- $tanh'$ is bounded from above by 1, approximating zero for $|x| > 3$

Combining this with a standard weight initialization scheme using independent Gaussian random variables, normalized to have zero mean and standard deviation of 1, we are multiplying the value of a gradient which is $< 1/4$ for σ and < 1 for $tanh$ with a weight matrix which is < 1, T times.

$$\frac{\partial h_t}{\partial h_{t-2}} = \underbrace{\sigma'(W_{hh}\sigma(W_{hh}h_{t-2}))}_{<1/4} \cdot \overbrace{W_{hh}}^{<1} \cdot \underbrace{\sigma'(W_{hh}h_{t-2}))}_{<1/4} \cdot \overbrace{W_{hh}}^{<1} \tag{4.2.10}$$

This effectively eliminates the gradient and the influence of the weights as we increase the number of time steps T. There are mitigation strategies but more commonly, the LSTM architecture outlined in Section 4.3 is used in practice.

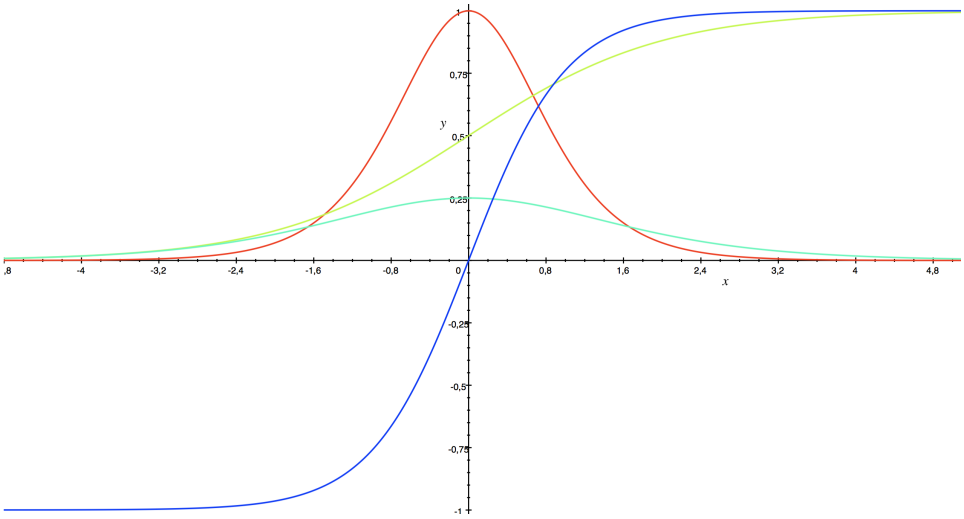


Figure 4: Activation functions σ (yellow) and $tanh$ (blue) and respective derivatives (green, red).

4.3 Long Short-Term Memory networks

Since its original formulation [21] Long Short-Term Memory (LSTM) networks have proven a scalable and effective solution to modelling complex, sequential data.

The LSTM architecture provides a solution to the vanishing gradient problem outlined in Section 4.2.2 by introducing an LSTM block where the state at time t is split into an internal state c and an output state h. As with the basic recurrent architecture, the update of these depends on the previous state and the current input but are then further controlled by a number of multiplicative gates as outlined in Section 4.3.2.

4.3.1 Naming convention and notation

However, before outlining the equations of the forward pass, we start by summarising notation and naming conventions used in the existing literature when describing LSTM networks. From here we define notation and naming conventions used throughout the rest of this thesis.

For all references, figures used for describing the architecture can be found in Appendix B and will be referenced throughout the section.

- The original paper [21] introduces the `memory cell` as a self-connected, linear `unit` j in combination with an `input gate unit` and an `output gate unit` (Figure 35).

  A `memory cell block` is then comprised of multiple `memory cells` sharing input and output gates (Figure 36).

- The book [17] agrees with the definitions of [21] and defines an `LSTM cell` to be comprised of a system of gating units and a block as containing multiple cells (Figure 31).

- Later papers [18, 19] adopt the notation of a block but uses it with the same definition as the `memory cell` of [21]. The block is referred to as an LSTM block (Figures 34 and 32) and consists of a single cell and its gate units. The cell within the block is thus defined without the input and output gate units[4] – what [21] refers to as j.

  One or more LSTM blocks are defined as the LSTM layer and the input and output of the block are named block input and block output respectively (Figure 33).

- Other papers [34] do not mention blocks, but instead refer to the LSTM architecture as hidden layer (Figure 38) consisting of one or more memory cells. Here, a memory cell also includes the gate units (Figure 37).

- Finally the open source software library TensorFlow [1] defines the parameter `num_units` as "the number of units in the LSTM cell"[5]. Applying the definitions of [19], the `num_units` parameter should be interpreted as the number of LSTM blocks within each LSTM layer and conversely, the LSTM cell in the TensorFlow definition should be interpreted the as the LSTM layer [19] consisting of `num_units` blocks.

In the rest of the thesis, we will adopt the naming convention of [19, 18] and the notation of the individual units from [19, 17] combined. The former is outlined in Table 4.3.2 below, while the latter is mapped to Equations 4.3.1 and 4.3.2 of the forward pass.

1. *LSTM block*: A unit comprised of a self connected cell and a number of multiplicative gates. Number depending on architecture.

2. *LSTM layer*: A unit consisting of one or more LSTM blocks.

4.3.2 Forward pass

The following outlines the forward pass of the basic LSTM architecture used throughout this thesis. Refer to Section 4.4 for a more thorough discussion on variants of the LSTM architecture.

Given N LSTM blocks and M inputs we have:

[4] The forget gate was first introduced with [16]

[5] The documentation also notes, that this definition differs from the consensus established in the literature.

1. $W_{zx}, W_{ix}, W_{fx}, W_{ox} \in \mathbb{R}^{NxM}$ as the weights on the input from the previous layer.
2. $W_{zh}, W_{ih}, W_{fh}, W_{oh} \in \mathbb{R}^{NxN}$ as the weights on the input from the previous hidden state.
3. $b_z, b_i, b_f, b_o \in \mathbb{R}^N$ as the bias weights

and can now define the forward pass of the LSTM layer for an input vector x_t at time t as:

$$\begin{aligned} i_t &= \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i) \\ f_t &= \sigma(W_{fh}h_{t-1} + W_{fx}x_t + b_f) \\ o_t &= \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o) \\ z_t &= \theta(W_{zh}h_{t-1} + W_{zx}x_t + b_z) \end{aligned} \tag{4.3.1}$$

$$\begin{aligned} c_t &= f_t \odot c_{t-1} + i_t \odot z_t \\ h_t &= o_t \odot \theta(c_t) \end{aligned} \tag{4.3.2}$$

where $\odot$ denotes element-wise multiplication of two vectors, σ the logistic sigmoid used for gate activation and θ a non-linear activation function used for input and output activation – usually the tanh. Further,

1. i is the input gate.
2. f is the forget gate introduced in [16]
3. o is the output gate
4. z is the input to the LSTM layer distributed to each of the LSTM blocks it contains. In the original paper, this is called y_{in}.
5. c is the internal cell state of the LSTM block
6. h is the value of the LSTM block at time t, equivalent to the value of the hidden state vector of the basic recurrent network outlined in Equation 4.2.2. In the original paper and [19] this is referred to as y_{out} and y_t respectively. We adopt the notation of [17].

The output h of the LSTM block is recurrently connected to the block input and all of the gates. A schematic of the equations is outlined in Figure 5.

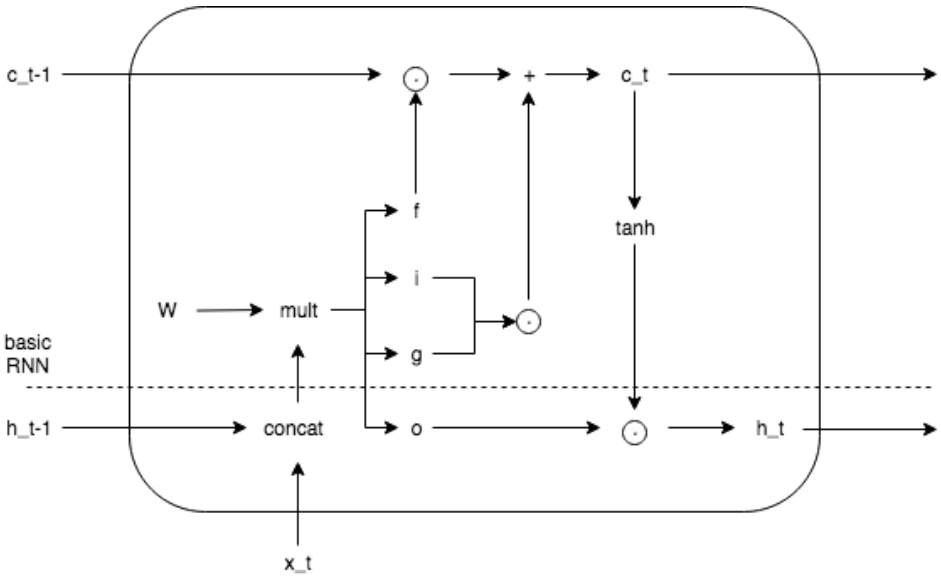


Figure 5: Schematic of the LSTM block, showing how the output h_t of the block at time t is computed from its value at time $t-1$, the cell state and the multiplicative gates.

4.3.3 Back propagation

The gates of the LSTM block thus each have their own set of weight values and we can compute gradients and update weights using them during the backward pass – just like we would in the basic recurrent neural network.

Deriving gradients and recursively applying them using the chain rule through the backpropagation algorithm is handled by the TensorFlow library using automatic differentiation [40]. Figure 6 shows computational graph built by the library, given the simple TensorFlow program below:

```
x = tf.Variable(1.0)
y = tf.square(x)
tf.train.GradientDescentOptimizer(0.01).minimize(y)
```

As highlighted, the square x^2 operation is decomposed into expressions consisting of at most one function call and its gradient $\frac{\partial x^2}{\partial x} = 2x$ is computed and put on the graph. These expression can then be applied using the chain rule during backpropagation.

To make the learning process more tractable; given a sequence of length n, the entire sequence is divided into multiple sequences with a fixed length T. Inputs of size T are then fed into the network and backpropagation is performed after each such input. The procedure is discussed in further

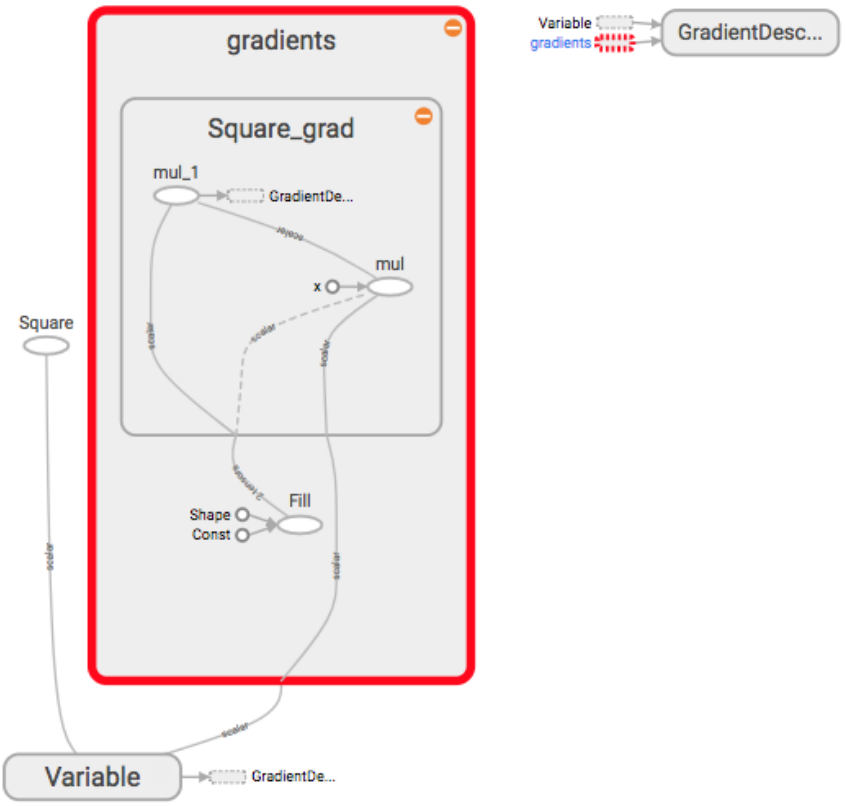


Figure 6: Forward pass of the automatic differentiation in TensorFlow. The expression x^2 is decomposed and the gradient $\frac{\partial x^2}{\partial x} = 2x$ is computed.

details in Section 5.5.1.1.

4.4 LSTM variants and the GRU

Numerous modifications to the initial LSTM block have been proposed since its introduction. Some papers [19, 29] attempt to back the intuition and experience gained by practitioners over time with experimental results by searching over huge number of changes and modifications to both LSTM architecture and hyper parameters. None of them find anything that performs substantially better than the basic LSTM setup defined in Section 4.3.2.

Sections 4.4.1 and 4.4.2 outlines two of the modifications tested in [19, 29] which will form the basis for our experiment alongside the basic RNN and the basic LSTM architecture. We choose these, since they are presented

as the most promising. A more general discussion of none-LSTM specific modifications to the network during training is outlined in Section 5.

4.4.1 Peepholes

Peepholes were introduced with [15]. Adding these to Equations 4.3.1, we get:

$$\begin{aligned} f_t &= \sigma(W_{fh}h_{t-1} + W_{fx}x_t + p_f \odot c_t + b_f) \\ o_t &= \sigma(W_{oh}h_{t-1} + W_{ox}x_t + p_o \odot c_{t-1} + b_o) \\ z_t &= \theta(W_{zh}h_{t-1} + W_{zx}x_t + p_z \odot c_{t-1} + b_z) \end{aligned} \tag{4.4.1}$$

where $p_f, p_o, p_z \in \mathbb{R}^N$ and $\odot$ denotes element-wise multiplication of two vectors.

Peepholes thus add a connection between the *input*, *output* and *forget* gates and the cell state c of the LSTM block. This allows the cell state to influence the gates directly, without first passing through the output gate o. The authors note that they intend to *"apply LSTM to music and rythm classification"*, suggesting that increase in predictive performance gained by adding peephole connections might be limited data with certain characteristics, not necessarily exhibited by financial time series.

4.4.2 Gated Recurrent Unit

The Gated Recurrent Unit (GRU) was introduced in [10] as a simplified variant of the LSTM cell promising both simpler implementation and faster compute times. The architecture combines the forget and input gates of Equation 4.3.1 and merges the internal and hidden cell states of equation 4.3.2 to the *reset* and *update* gates r_t and z_t and the output states h_t given in Equation 4.4.2.

$$\begin{aligned} r_t &= \sigma(W_{rh}h_{t-1} + W_{rx}x_t + b_r) \\ z_t &= \sigma(W_{zh}h_{t-1} + W_{zx}x_t + b_z) \\ \tilde{h_t} &= \theta(W_x x_t + W_h[r_t \odot h_{t-1}] + b) \\ h_t &= z_t h_{t-1} + (1 - z_t)\tilde{h_t} \end{aligned} \tag{4.4.2}$$

The output states h_t are thus controlled only by two gates, r_t and z_t. θ is a non-linear activation function and again $\odot$ denotes element-wise multiplication. As with the LSTM equations, given N GRU blocks and M inputs we have:

1. $W_{zx}, W_{rx}, W_x \in \mathbb{R}^{NxM}$

2. $W_{zh}, W_{rh}, W_h \in \mathbb{R}^{NxN}$

3. $b_z, b_r, b \in \mathbb{R}^N$

5 Experiment setup

In the following section we outline the details for obtaining the predicted values and results presented in Section 6.

5.1 Hardware and software

All data handling and processing is conducted in Python 2.7 (3.1 GHz Intel Core i5, 16GB RAM) with packages *numpy*[6] and *pandas*[7]. For implementation of LSTM networks the open source framework TensorFlow v1.5 [1] was used. For calculation of technical indicators we rely on the *ta-lib*[8] package. And finally, for performance evaluation we rely on the *sklearn*[9] and *scipy*[10] packages.

Appendix C outlines some practical details and notes on actual LSTM network implementation in TensorFlow and will be referenced throughout the section.

5.2 Prediction approach

The selection of data sets, input variables, time frames and study periods for model prediction follows the procedure described in [5]. In particular, the entire data set is split into subsets and for each subset, further split into 3 none-overlapping time periods:

1. A training set consisting of data points from the past two year period.

2. A validation set consisting of data points from the 3 month period directly following the training period.

3. A test set consisting of data points from the 12 month period directly following the validation period.

After fitting the model, a network with parameters from the parameter combination with the lowest generalization error is trained on the training and validation sets combined before being used for carrying out the final prediction on the test set. The results are presented in Section 6. In total, the process is repeated over 6 years, predicting yearly performance as

[6] http://www.numpy.org/
[7] https://pandas.pydata.org/
[8] https://www.ta-lib.org/
[9] http://scikit-learn.org/stable/documentation.html
[10] https://www.scipy.org/

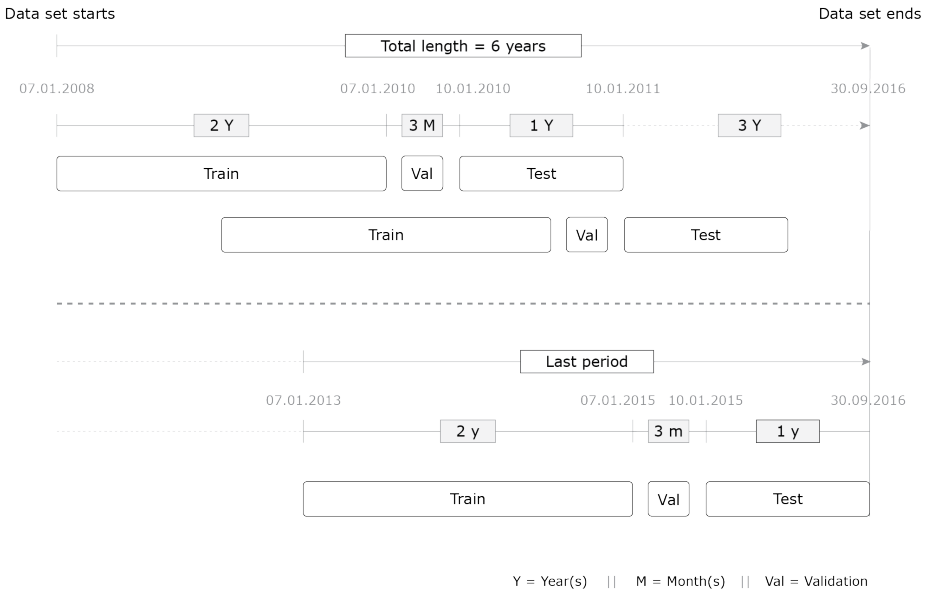


Figure 7: Prediction periods. For each year over a 6 year period the data set is split into training, validation and test set.

outlined in Figure 7. For all data sets, one data point per day is recorded. Hyperparameter and model selection are elaborated on in Section 5.5.

5.2.1 Sample selection and input variables

We select three major indices to assess the performance of the proposed models. Those include:

1. The American stock market index Standard & Poor's 500 (SP500): an index based on a weighted average of the market capitalizations of the 500 largest companies with common stock listed on American exchanges.

2. The Hang Seng index (HSI): equivalent to the SP500 but comprised of the largest companies on the Hong Kong stock market exchanges.

3. The Nikkei 225 (N225): a price weighted index of the Tokyo Stock Exchange.

The selection of different indices allows us to represent potentially different market conditions and thus further stress the robustness of our proposed models. All are listed in Table 1 with closing hours.

For each index, two types of input variable categories are used:

Index	Country	Closing time (GMT)
Hang Seng	Hong Kong	0900
Nikkei 225	Japan	0700
SP500	US	2100

Table 1: Stock indices and closing times

Name	Description
Daily trading data	
Open	Price at opening of the exchange
Close	Price at closing of the exchange
High	Highest price during the daily opening
Low	Lowest price during the daily opening
Volume	Trading volume
Technical indicators	
MACD	Moving Average Convergence/Divergence
CCI	Commodity Channel Index
ATR	Average True Range
BBANDS	Bollinger Bands
EMA	Exponential Moving Average
MA	All Moving Average
ROC	Rate of change : ((price/prevPrice)-1)*100
WILLR	Williams' %R

Table 2: Input variables grouped by type.

- Historical trading data. This includes daily open, close, high and low prices as well as trading volume of the index.
- Technical indicators commonly used in the industry.

Table 2 briefly summarizes actual variables by name and definition[11]. Refer to Section 3 for further elaboration and equations.

5.2.2 Data collection

All data sets are collected using the publicly available sources provided at Yahoo Finance[12]. In total, data points from January 1st 2008 - October 31st 2017 have been collected.

[11] Abbreviation and short description based on *ta-lib* documentation: http://ta-lib.org/function.html

[12] https://finance.yahoo.com/quote/[TICKER]/history?period1=[FROM] &period2=[TO]&interval=1d&filter=history&frequency=1d

5.2.3 Preprocessing

All input features of the training set are normalized to have zero mean and unit variance. The target vector is likewise normalized to have zero mean and unit variance, using the mean and standard deviation of the training set. Upon prediction, the forecasted values are transformed back to original scale.

5.3 Performance measurement

For performance measurement, we limit our focus to predictive accuracy and do not address the potential for converting this predictive accuracy to an actual trading strategy and an according profitability performance measure.

The predictive accuracy for each RNN model is reported using the R^2 [46] measure as defined in Equation 5.3.2. For benchmarks against existing literature we are using the R [47] measure defined in Equation 5.3.3[13]. Both are widely used to provide information on the goodness of a fit of a model, independent on feature scaling. A measure of 1 indicates a perfect fit. For model fitting and parameter optimization we rely on the mean squared error outlined in Equation 5.3.4.

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{5.3.1}$$

$$\mathbf{R^2} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5.3.2}$$

$$\mathbf{R} = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(\hat{y}_i - \hat{\bar{y}})}{\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2(\hat{y}_i - \hat{\bar{y}})^2}} \tag{5.3.3}$$

$$\mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5.3.4}$$

In all equations y represents the expected value, $\hat{y}$ the predicted and n the length of the prediction period.

It can be shown from Equations 5.3.2 and 5.3.4 that the R^2 measure can be interpreted as a normalized version of the mean squared error. We use it here, because interpretation does not depend on the scale of the data.

[13] We have to use two different measures as [5] reports R and RiskButler R^2

Parameter	Values
Learning rate	[0.005, 0.001]
Number of layers	[1, 2]
Number of blocks	[64, 128, 256]
Truncated length	[20, 50, None]
Forget gate bias	[0.5, 1.0]
Dropout	[0.0, 0.5]
Gradient norm threshold	[None, 0.5, 1.0]

Table 3: Hyperparameter space for grid search

5.4 Model selection

To evaluate performance of the RNN models, the following steps are taken:

1. First, we build a model hyperparameter optimization pipeline according to established and documented optimization techniques. The approach is described in Section 5.5.

2. Second, we compare the basic LSTM model to its peephole counterpart, the basic RNN and the GRU architecture by tuning each model and reporting both measures on accuracy, hyperparameters and compute time.

3. The results are then benchmarked against results from existing literature [5] before finally, we report benchmark measures from the stochastic models described in Section 3.1.3.

For each proposed model, feeding all input variables described in Table 2 is compared to feeding only lagged closing prices. The prediction procedure is repeated 5 times and the results are averaged and reported in Section 6. These last two steps are done in addition to the procedure outlined in [5] and allows us to both minimize the possibility of one particular random initialization favoring one model over another and to reason about statistical significance when comparing the RNN models.

5.5 Model optimization

This section outlines the process for selecting the optimal parameter configuration of the recurrent neural networks.

In total 7 hyperparameters are tuned by performing a grid search over the hyperparameter space outlined in Table 3. If nothing else is described, ranges for the parameter space were chosen by manual trial and error. Results and final network configurations are presented in Section 6.

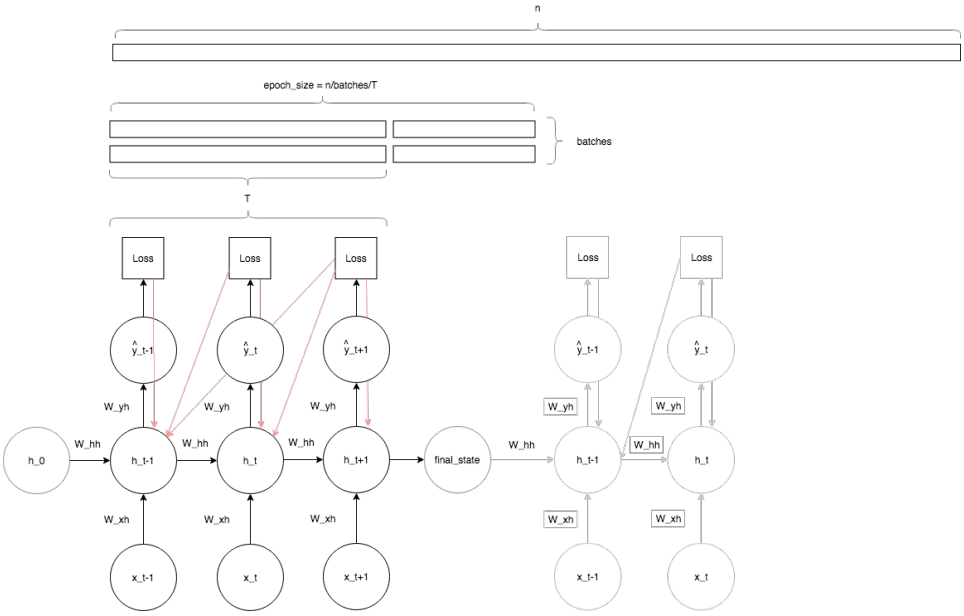


Figure 8: Visualization of the truncated backpropagation for a sequence of length $n = 10$ and a fixed maxium $T = 3$

5.5.1 Optimization

5.5.1.1 Truncated backpropagation The LSTM architectures address the problem of vanishing gradients, clipping gradients as described in Section 5.5.1.5 addresses the problem of exploding gradients; however, training LSTMs can still be computationally expensive as the number of time steps increases.

To remedy this, given a sequence of length n, we truncate the length of the sequences fed into the network to a fixed maximum T and perform our backward pass on the sub-sequence as illustrated in Figure 8. TensorFlow implementation is enclosed in Appendix C.

5.5.1.2 Early stopping As suggested in [6] we employ *early stopping* by monitoring the validation loss as training progresses for every N epochs. The stopping condition is implemented as a *patience* of 10 and a *tolerance* of 10^{-3}.

5.5.1.3 Adaptive learning rate For all experiments, the Adam optimization algorithm [30] was utilized. The algorithm distinguishes itself from others by computing individual adaptive learning rates for each parameter by calculating estimates of first and second moments of the gradients. Only the learning rate parameter α was optimized during grid search.

5.5.1.4 Decaying learning rate

The Adam optimization algorithm associates an individual learning rate to every parameter in the network. However, as this parameter is limited from above by an initial learning rate, we can 1) help converge faster by starting out with a higher learning rate and 2) reduce final loss by decreasing limit when loss with the previously associated learning rate does not provide more accuracy.

The decayed learning rate is computed by Equation 5.5.1, with a decay rate of 0.95 and *decay_steps* equal to the truncated sequence length.

$$decayed\_lr = lr * decay\_rate^{(global\_step/\,decay\_steps)} \tag{5.5.1}$$

5.5.1.5 Gradient Clipping

As noted in Section 4.2.2 the LSTM network will potentially suffer from exploding gradients. To mitigate this, [39] proposes clipping the gradients if reaching a certain threshold by the algorithm given in Algorithm 2:

$\hat{g} = \frac{\partial \epsilon}{\partial \theta}$
if $||\hat{g}|| > threshold$ **then**
  $\hat{g} = \frac{threshold}{||\hat{g}||}\hat{g}$
end

Algorithm 2: Pseudo code for calculating the global norm and thresholding gradients.

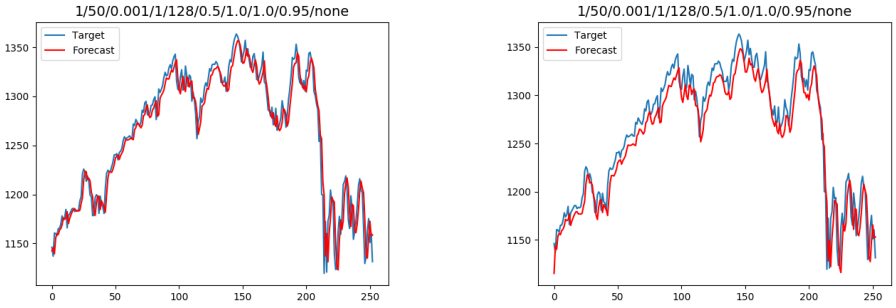
The threshold is introduced as a hyperparameter in Table 3. Values are chosen as suggested in the paper by looking at *"statistics on the average norm over a sufficiently large number of updates"*. Section 6.1.2 presents and discusses the results from the experiment.

5.5.2 Model parameters

5.5.2.1 Preserving hidden state

A potential issue needed to be addressed when working on truncated sequences is the need to preserve the hidden state of the network between each truncated sequence. We do this by passing the resulting hidden state of a truncated sequence as the initial hidden state of the next truncated sequence. This way, we eliminate potentially large loss terms for the initial time steps.

The same issue and solution applies for passing the last hidden state of the training pass as the initial state of the out-of-sample test pass. Figures 9a and 9b visualize the problem. The default TensorFlow implementation will reset the hidden state to zeros after each truncated sequence. See Appendix C for notes on actual implementation. The section also includes Figure 39 with a visualization of the problem applied to more human readable sine waves time series.



(a) Preserving (b) Without preserving

Figure 9: Visualization of potentially large loss term on initial time steps. In Figure 9a, we pass on the resulting state of the training pass. Figure 9b initializes the hidden state to zeros.

5.5.2.2 Variable sequence length Another issue with truncated back-propagation is that the maximum sequence length T might not be a divisor of the total sequence length n. Thus, we need to be able to pass sequences of variable length to the recurrent network. See Appendix C for notes on actual implementation.

5.5.2.3 Dropout Dropout [44] has been widely adopted as a state of the art regularization method for traditional feed forward networks. However, empirical research suggests that traditional dropout tends not to work well when applied to recurrent networks.

Instead, we rely on recurrent dropout as suggested by [48]. Modifying Equation 4.3.1 we get the following regularization scheme:

$$\begin{aligned} i_t &= \sigma(W_{ih}h_{t-1} + W_{ix}D(x_t) + b_i) \\ f_t &= \sigma(W_{fh}h_{t-1} + W_{fx}D(x_t) + b_f) \\ o_t &= \sigma(W_{oh}h_{t-1} + W_{ox}D(x_t) + b_o) \\ z_t &= \theta(W_{zh}h_{t-1} + W_{zx}D(x_t) + b_z) \end{aligned} \tag{5.5.2}$$

where the D is the dropout function defined as:

$$D(x) = \begin{cases} x * (1/p) & \text{with probability } p \\ 0 & \text{otherwise} \end{cases} \tag{5.5.3}$$

Notice how we are scaling up inputs that are not dropped. This is required, since we only apply the dropout function D in the training pass and keep all inputs in the validation and final test passes. Hence, to get

the output in the validation and test passes to the same scale as the expected output in the training pass, we have to scale. Actual implementation handled by the TensorFlow library.

The dropout function thus sets a random subset of its argument to zero, or scales it to compensate for the dropout. This addition forces the network to have redundant representation and not rely solely on one specific set of features, since they might not be available.
It should be noted that a great variety of dropout implementation exist; Some modifying the cell state [38] others the recurrent connections [14].

5.5.2.4 Forget gate bias Increasing or decreasing the initial forget gate bias of Equation 4.3.1 can help reduce or increase the influence of the previous hidden state on the current hidden state during the beginning of the training. In TensorFlow default value is 1.

5.5.3 Architecture

5.5.3.1 Stacking multiple RNNs It is possible to increase the depth of the RNN model by passing the output of a RNN layer as the input to a subsequent RNN layer. Some [43] report improved result by increasing the number of RNN layers but currently no good rule of thumb seems to exist. In our case, it did not increase performance.

6 Results

This section presents the empirical results of the experiment. First, the predictive performance the RNN architectures is reported. For each architecture the results are grouped by data set and time frame and a visualization of corresponding predictions and targets is enclosed along the naive benchmark measure of always predicting the value at time $t-1$ at time t (`t-1`). The results are then benchmarked against results from [5] and statistical significance is addressed. Finally, the GRU model is benchmarked against results from the stochastic models of the RiskButler portfolio.

In Appendix D a link is enclosed to a csv file containing predictions, measures on predictive accuracy and hyperparameter configurations for each run and benchmark model in the results presented.

6.1 Model comparison

For each model, we report 1) the median R^2 score grouped by data set and time period and 2) the relative compute time, calculated as the percentage of the sum of the total compute time in minutes grouped by data set and time period. Tables 10a and 10b present the results, where:

- `basic_rnn_cell` denotes the basic RNN architecture described in Section 4.2
- `basic_lstm_cell` the LSTM architecture from Section 4.3
- `peep_lstm_cell` the peephole architecture from 4.4.1
- `gru_cell` the GRU architecture described in 4.4.2

Further, each model is compared individually to the `basic_lstm_cell` in Figure 11.

6.1.0.1 Predictive performance As can be seen, the `basic_lstm_cell` outperforms the `basic_rnn_cell` in 16 of 18 cases, the `basic_lstm_cell` is almost tied with `peep_lstm_cell` architecture with 10 of 18 in favor of the `peep_lstm_cell`, the `gru_cell` outperforms the `basic_lstm_cell` in 12 of 18 cases, and is tied with the `peep_lstm_cell` when compared one on one.

6.1.0.2 Computational cost The `gru_cell` performs at a decrease in computational cost[14] while the `peep_lstm_cell` performs at an increase in same when compared to the `basic_lstm_cell`.

Section 6.1.1 elaborates on the significance of results and summarises conclusions on the trade-off between increased predictive performance and computational cost.

6.1.1 Statistical significance

A consequence of starting from a random initialization is that each experiment must be repeated several times to determine significance. To address this, each experiment is conducted 5 times as described in Section 5.3 and resulting statistical summaries, box plots and histograms are presented in Table 13a and Figures 13b and 13c.

From both summaries and box plots, it is clear that 1) the models exhibit different statistical characteristics and that 2) these differ from data set to data set. The `basic_rnn_cell` clearly is more unstable overall with a minimum of 0.602520 and maximum of 0.971968 yielding a standard deviation of 0.109333 on the SP500 data set in the most extreme case.

The forecasting ability of all models across the 6 year period thus varies from data set to data set, but the Nikkei index generally yields higher fluctuations than the SP500 and the HSI indices. An in depth interpretation of the different market conditions influence on the forecasting ability lies beyond the

[14]The computational calculation does not take into account hyperparameter configuration such as number of blocks.

MEDIAN of r2		model				
target_column	start_date_test	basic_lstm_cell	basic_rnn_cell	gru_cell	peep_lstm_cell	t-1
hsi_close	2010-10-01	0.9601	0.9571	0.9592	0.9604	0.9660
	2011-10-01	0.9324	0.9315	0.9333	0.9319	0.9329
	2012-10-01	0.9343	0.9295	0.9356	0.9297	0.9374
	2013-10-01	0.9461	0.9521	0.9455	0.9527	0.9542
	2014-10-01	0.9681	0.9659	0.9734	0.9718	0.9742
	2015-10-01	0.9638	0.9523	0.9628	0.9578	0.9644
hsi_close Total		0.9505	0.9517	0.9510	0.9528	0.9593
n225_close	2010-10-01	0.9445	0.9405	0.9438	0.9445	0.9440
	2011-10-01	0.9571	0.9445	0.9586	0.9165	0.9606
	2012-10-01	0.9835	0.7663	0.9667	0.9046	0.9889
	2013-10-01	0.8823	0.7931	0.8965	0.9067	0.9194
	2014-10-01	0.9437	0.9158	0.9547	0.9636	0.9754
	2015-10-01	0.9493	0.9446	0.9496	0.9492	0.9491
n225_close Total		0.9491	0.9336	0.9494	0.9424	0.9548
sp500_close	2010-10-01	0.9463	0.8000	0.9393	0.9428	0.9456
	2011-10-01	0.9645	0.9577	0.9620	0.9613	0.9644
	2012-10-01	0.9798	0.9375	0.9828	0.9870	0.9875
	2013-10-01	0.9209	0.7161	0.9787	0.9493	0.9800
	2014-10-01	0.8414	0.8846	0.8981	0.9004	0.9106
	2015-10-01	0.9499	0.9492	0.9516	0.9527	0.9529
sp500_close Total		0.9472	0.9015	0.9525	0.9526	0.9587

(a) RNN model comparison by R^2 measure.

SUM of compute_time		model			
target_column	start_date_test	basic_lstm_cell	basic_rnn_cell	gru_cell	peep_lstm_cell
hsi_close	2010-10-01	0.2121	0.2019	0.3147	0.2713
	2011-10-01	0.2644	0.1694	0.2641	0.3021
	2012-10-01	0.2765	0.1012	0.2560	0.3663
	2013-10-01	0.3362	0.1786	0.1974	0.2878
	2014-10-01	0.4992	0.0404	0.3100	0.1504
	2015-10-01	0.2776	0.1832	0.1804	0.3589
hsi_close Total		0.3029	0.1514	0.2460	0.2997
n225_close	2010-10-01	0.3213	0.1728	0.3275	0.1784
	2011-10-01	0.2870	0.0914	0.2661	0.3556
	2012-10-01	0.1885	0.1283	0.3914	0.2918
	2013-10-01	0.2914	0.2065	0.2444	0.2577
	2014-10-01	0.3739	0.1743	0.2538	0.1980
	2015-10-01	0.2997	0.0996	0.2724	0.3282
n225_close Total		0.2939	0.1442	0.2884	0.2735
sp500_close	2010-10-01	0.3159	0.0517	0.3253	0.3071
	2011-10-01	0.2557	0.0934	0.1767	0.4743
	2012-10-01	0.3235	0.0764	0.2568	0.3433
	2013-10-01	0.2268	0.2086	0.2474	0.3172
	2014-10-01	0.2098	0.1760	0.2155	0.3988
	2015-10-01	0.2436	0.1232	0.2567	0.3765
sp500_close Total		0.2595	0.1255	0.2424	0.3726

(b) RNN model comparison by relative compute time calculated as the percentage of the sum of the total compute time in minutes grouped by data set and time period.

MEDIAN of r2		model	
target_column	start_date_test	basic_lstm_cell	peep_lstm_cell
hsi_close	2010-10-01	0.9601	0.9604
	2011-10-01	0.9324	0.9319
	2012-10-01	0.9343	0.9297
	2013-10-01	0.9461	0.9527
	2014-10-01	0.9681	0.9718
	2015-10-01	0.9638	0.9578
hsi_close Total		0.9505	0.9528
n225_close	2010-10-01	0.9445	0.9445
	2011-10-01	0.9571	0.9165
	2012-10-01	0.9835	0.9046
	2013-10-01	0.8823	0.9067
	2014-10-01	0.9437	0.9636
	2015-10-01	0.9493	0.9492
n225_close Total		0.9491	0.9424
sp500_close	2010-10-01	0.9463	0.9428
	2011-10-01	0.9645	0.9613
	2012-10-01	0.9798	0.9870
	2013-10-01	0.9209	0.9493
	2014-10-01	0.8414	0.9004
	2015-10-01	0.9499	0.9527
sp500_close Total		0.9472	0.9526

(a) `basic_lstm_cell/peep_lstm_cell`

MEDIAN of r2		model	
target_column	start_date_test	basic_lstm_cell	basic_rnn_cell
hsi_close	2010-10-01	0.9601	0.9571
	2011-10-01	0.9324	0.9315
	2012-10-01	0.9343	0.9295
	2013-10-01	0.9461	0.9521
	2014-10-01	0.9681	0.9659
	2015-10-01	0.9638	0.9523
hsi_close Total		0.9505	0.9517
n225_close	2010-10-01	0.9445	0.9405
	2011-10-01	0.9571	0.9445
	2012-10-01	0.9835	0.7663
	2013-10-01	0.8823	0.7931
	2014-10-01	0.9437	0.9158
	2015-10-01	0.9493	0.9446
n225_close Total		0.9491	0.9336
sp500_close	2010-10-01	0.9463	0.8000
	2011-10-01	0.9645	0.9577
	2012-10-01	0.9798	0.9375
	2013-10-01	0.9209	0.7161
	2014-10-01	0.8414	0.8846
	2015-10-01	0.9499	0.9492
sp500_close Total		0.9472	0.9015

(b) `basic_lstm_cell/basic_rnn_cell`

MEDIAN of r2		model	
target_column	start_date_test	basic_lstm_cell	gru_cell
hsi_close	2010-10-01	0.9601	0.9592
	2011-10-01	0.9324	0.9333
	2012-10-01	0.9343	0.9356
	2013-10-01	0.9461	0.9455
	2014-10-01	0.9681	0.9734
	2015-10-01	0.9638	0.9628
hsi_close Total		0.9505	0.9510
n225_close	2010-10-01	0.9445	0.9438
	2011-10-01	0.9571	0.9586
	2012-10-01	0.9835	0.9667
	2013-10-01	0.8823	0.8965
	2014-10-01	0.9437	0.9547
	2015-10-01	0.9493	0.9496
n225_close Total		0.9491	0.9494
sp500_close	2010-10-01	0.9463	0.9393
	2011-10-01	0.9645	0.9620
	2012-10-01	0.9798	0.9828
	2013-10-01	0.9209	0.9787
	2014-10-01	0.8414	0.8981
	2015-10-01	0.9499	0.9516
sp500_close Total		0.9472	0.9525

(c) `basic_lstm_cell/gru_cell`

MEDIAN of r2		model	
target_column	start_date_test	gru_cell	peep_lstm_cell
hsi_close	2010-10-01	0.9592	0.9604
	2011-10-01	0.9333	0.9319
	2012-10-01	0.9356	0.9297
	2013-10-01	0.9455	0.9527
	2014-10-01	0.9734	0.9718
	2015-10-01	0.9628	0.9578
hsi_close Total		0.9510	0.9528
n225_close	2010-10-01	0.9438	0.9445
	2011-10-01	0.9586	0.9165
	2012-10-01	0.9667	0.9046
	2013-10-01	0.8965	0.9067
	2014-10-01	0.9547	0.9636
	2015-10-01	0.9496	0.9492
n225_close Total		0.9494	0.9424
sp500_close	2010-10-01	0.9393	0.9428
	2011-10-01	0.9620	0.9613
	2012-10-01	0.9828	0.9870
	2013-10-01	0.9787	0.9493
	2014-10-01	0.8981	0.9004
	2015-10-01	0.9516	0.9527
sp500_close Total		0.9525	0.9526

(d) `gru_cell/peep_lstm_cell`

Figure 11: RNN model comparison by R^2 measure

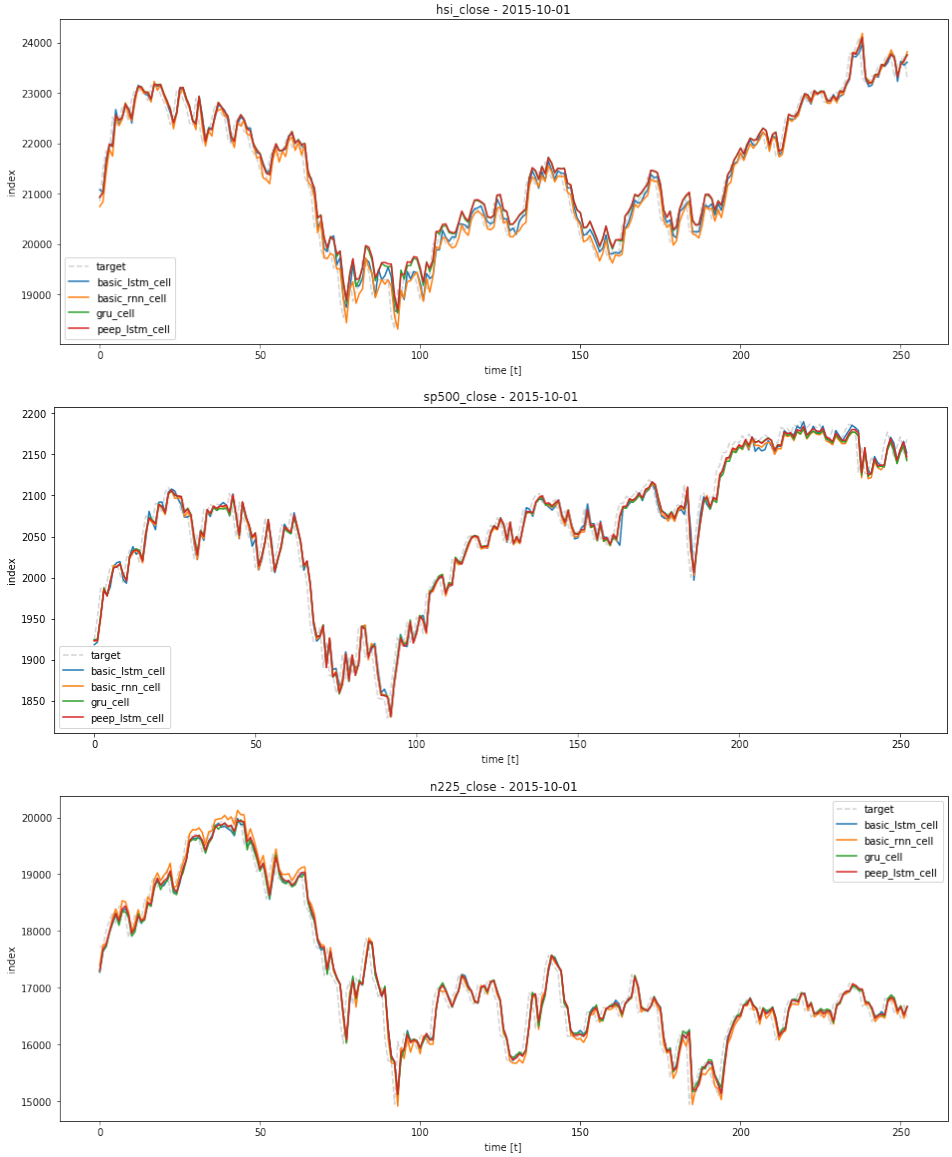
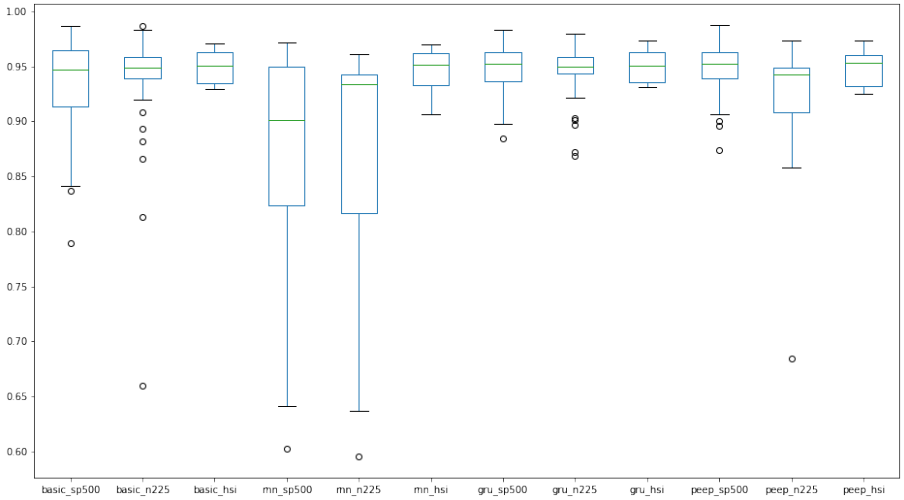


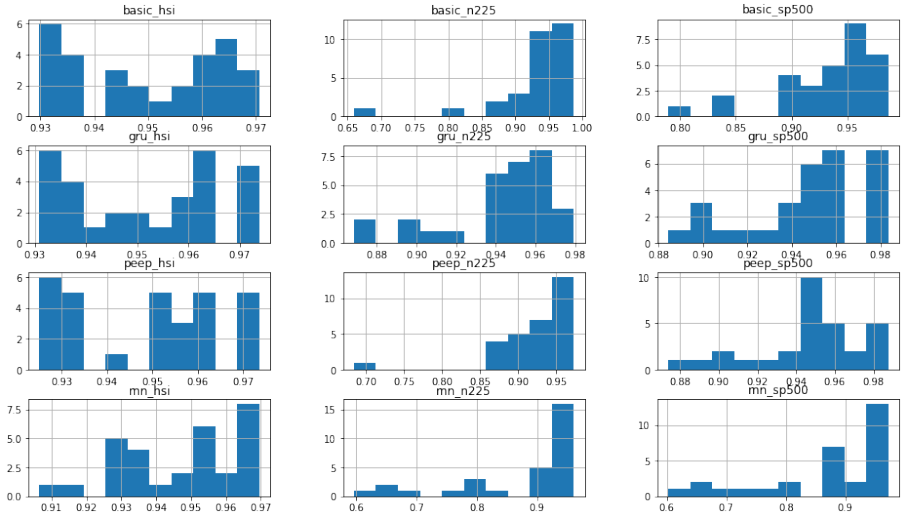
Figure 12: Target and predictions for each of the four models grouped by stock index with prediction start 10-01-2015 and end 09-31-2016. Plots of all results are included in Appendix D.

	basic_sp500	basic_n225	basic_hsi	rnn_sp500	rnn_n225	rnn_hsi	gru_sp500	gru_n225	gru_hsi	peep_sp500	peep_n225	peep_hsi
count	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000
mean	0.934163	0.931504	0.949811	0.869271	0.874386	0.946519	0.948035	0.942649	0.951167	0.948845	0.921710	0.948868
std	0.046623	0.063153	0.013950	0.109333	0.106807	0.016857	0.027869	0.028065	0.014743	0.028669	0.053542	0.016372
min	0.789550	0.659559	0.929818	0.602520	0.595832	0.906563	0.884478	0.868414	0.930891	0.873954	0.684479	0.925082
25%	0.913657	0.939500	0.934275	0.824114	0.816687	0.932787	0.936569	0.943282	0.935707	0.939277	0.908153	0.932010
50%	0.947197	0.949073	0.950514	0.901535	0.933621	0.951669	0.952531	0.949450	0.950966	0.952646	0.942435	0.952773
75%	0.964567	0.958114	0.962982	0.949275	0.943044	0.962167	0.963048	0.958822	0.962840	0.963072	0.948947	0.960307
max	0.986687	0.987097	0.970676	0.971968	0.961314	0.969607	0.983533	0.979379	0.973820	0.987654	0.973374	0.973573

(a) Statistical summaries grouped by model and data set



(b) Box plots grouped by model and data set.



(c) Histograms grouped by model and data set.

scope of this thesis, but is does seem to confirm the hypothesis, that lesser developed markets exhibit more recognizable patterns than more developed markets, where the trading volume is higher and the price will more likely already embed all information available. Section 7.5 elaborates on the possibilities for exploiting this and determining other influential factors such as industry and macro economic trends.

6.1.1.1 Mann-Whitney U test A two-sided Mann-Whitney U test is performed to determine significance. The null hypothesis for the test is that the two groups have the same distribution while the alternative hypothesis is that one group has either larger or smaller values than the other. We choose the Mann-Whiney U test, as predictions from neither model follow the normal distribution and no relationship between predictions from each group or within each group exists.

The probability values are presented in Table 4. Here, the `basic_rnn_cell`, `gru_cell` and `peep_lstm_cell` models are compared to the `basic_lstm_cell`, grouped by data set. 2 things stand out:

- At a significance level of 0.05 we cannot tell neither the `gru_cell` nor the `peep_lstm_cell` from the `basic_lstm_cell`.
- We can, however, make a significant distinction between the `basic_lstm_cell` and the `basic_rnn_cell` and the `gru_cell` and the `basic_rnn_cell` respectively on both the SP500 and the Nikkei data sets.

Combined with the statistical summaries of Table 13a and overall gain in predictive accuracy in Table 10a these findings confirm the promises of Sections 4.4.2 and 4.4.1 respectively and reestablish what other empirical studies [19] have already confirmed:

- Both Long Short-Term Memory and GRU architectures do indeed increase performance at a significant level when compared to the basic RNN architecture.
- Modifications beyond the basic LSTM architecture do not result in any significant increase. In particular; adding peephole connections increases complexity of the network by 3 weights per LSTM block, and computational difficulty accordingly, with no significant performance gain.
- The relatively higher computational cost of the basic LSTM and the peephole LSTM architectures does not directly translate to superior performance when compared to the GRU architecture.

	basic_lstm_cell$_{\{dataset\}}$	gru_cell$_{\{dataset\}}$
basic_rnn_cell$_{sp500}$	0.0053220778	0.0003367901
basic_rnn_cell$_{hsi}$	0.5011436676	0.3183042275
basic_rnn_cell$_{n225}$	0.0002838867	0.0000814648
gru_cell$_{sp500}$	0.3870997777	-
gru_cell$_{hsi}$	0.7505872315	-
gru_cell$_{n225}$	0.8649937062	-
peep_lstm_cell$_{sp500}$	0.3402884651	0.8187456535
peep_lstm_cell$_{hsi}$	0.4733465573	0.2518808247
peep_lstm_cell$_{n225}$	0.0984134995	0.0751365999

Table 4: p values of two-sided Mann-Whitney U test, comparing predictions to the basic LSTM architecture grouped by data set

Further, from the standard deviations of Table 13a the GRU architecture yields the most stable results across runs. We hence conclude this Section with the take-away, that the GRU architecture is the architecture of choice amongst the 4 compared.

6.1.2 Hyperparameter configuration

Before benchmarking against the existing literature and the industry standard, optimal hyperparameter configurations are averaged and presented, grouped by model, data set and time frame in Tables 16 and 17.

6.1.2.1 Dropout Applying dropout improved out-of-sample performance in 17 of 18 cases for the basic_rnn_cell and 9, 10 and 12 cases for the gru_cell, peep_lstm_cell and basic_lstm_cell respectively. This may suggest, that the network is still sensitive to the weights of specific single units, that when dropped out decrease performance. An issue we address in Section 7.

6.1.2.2 Stacking RNNs In no case did increasing number of recurrent layers increase predictive performance.

6.1.2.3 Gradient clipping Clipping gradients was applied to some extend for all models. Figure 15 visualizes learning rates and gradient norms for an example run. On the left side, gradients have not been scaled and on the right, gradients exceeding the usual range have been thresholded. Otherwise optimal hyperparameters are chosen in both cases. As can be seen, the clipping helps both faster convergence and overall lower generalization error.

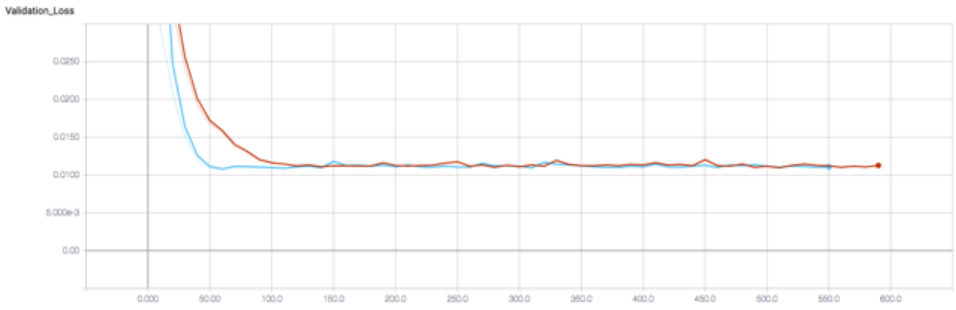


Figure 14: Learning rates with (blue) and without (red) clipping

6.1.2.4 Learning rate decay Decaying learning rate was applied to for all models.

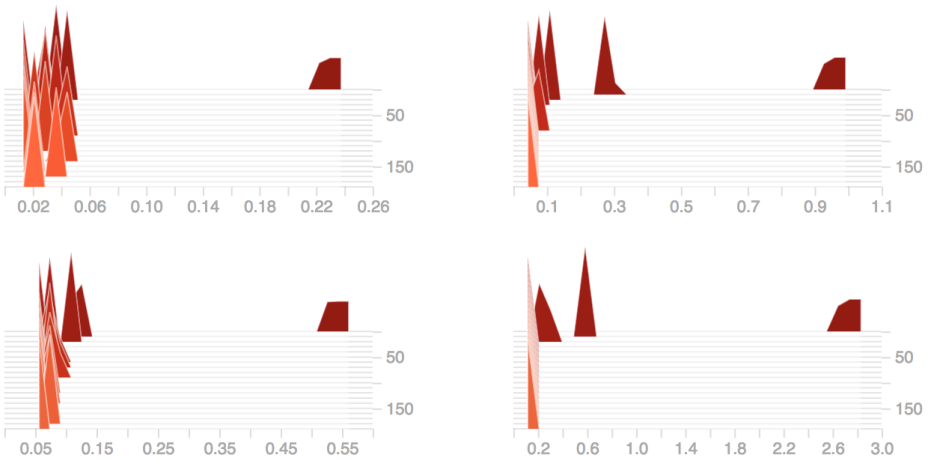
6.1.2.5 Sequence length A truncated sequence length of 50 performed best across models.

6.1.2.6 Number of blocks For all models, between 64 and 128 blocks seemed to perform the best.

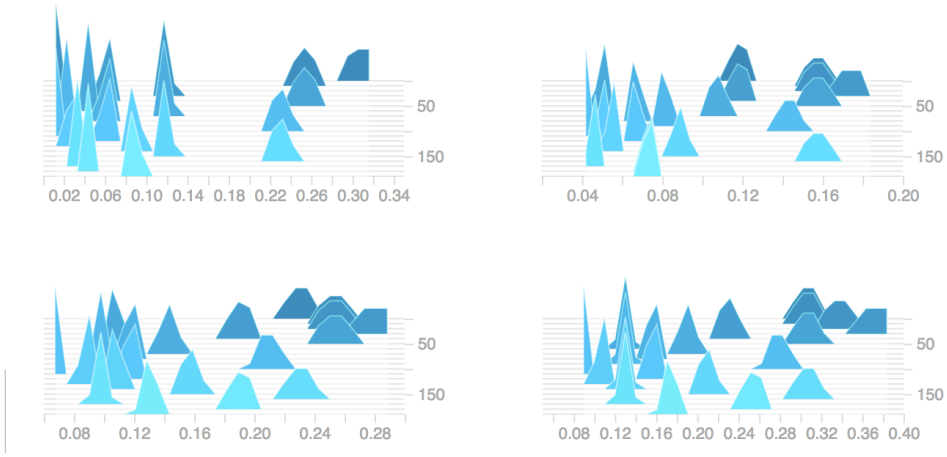
6.1.2.7 Forget gate bias For the LSTM models, in no case did changing the initial forget gate bias from the default increase performance.

6.1.2.8 Early stopping Learning rates without the early stopping mechanism employed for both truncated and full sequences with otherwise optimal hyperparameter configurations are plotted in Figure 18. As can be seen, we reach global minimum around 120 and 310 epochs respectively but from here neither do seem to improve performance nor be overfitting. This suggests that a deeper model might be able to extract more detailed patterns. Figure 19 visualizes training and validation errors for various combinations of increased number of layers and RNN blocks. As can be seen, neither seem to increase overall generalization performance. Refer to Section 7.5 more detailed suggestions on further improvements in this area.

6.1.2.9 Input variables For all models, feeding only lagged closing prices to the network provided the best results. With the application of dropout this is somewhat surprising, as we would expect the network to be able to learn and adapt weights to feature vectors of no or little relevance. Hence, this further increases our suspicion from above, that a deeper model might be able to extract more information.



(a) Without clipping



(b) Clipping gradients at 0.5

Figure 15: Gradient norms along the x-axis and the current epoch on the z-axis. In Figure 15b we threshold the gradient at 0.5, while in Figure 15a no clipping is performed.

		model	Values					
		basic_rnn_cell						
target_column	start_date_test	dropout	gradient_norm_threshold	decay_rate	hidden_size	sequence_length	learning_rate	epochs
hsi_close	2010-10-01	0.5	0	0.95	64	50	0.001	80
	2011-10-01	0.5	0.5	0.95	128	50	0.001	70
	2012-10-01	0.5	0.5	0.95	64	20	0.001	50
	2013-10-01	0.5	0	0.95	128	20	0.001	60
	2014-10-01	0	0.5	0.95	128	20	0.001	10
	2015-10-01	0.5	0	0.95	128	20	0.001	100
hsi_close Total		0.4166666667	0.25	0.95	106.6666667	30	0.001	61.66666667
n225_close	2010-10-01	0.5	0	0.95	64	50	0.001	60
	2011-10-01	0	0	0.95	64	20	0.001	40
	2012-10-01	0.5	0	0.95	64	50	0.001	40
	2013-10-01	0.5	0	0.95	64	20	0.001	90
	2014-10-01	0.5	0.5	0.95	64	20	0.001	70
	2015-10-01	0.5	0	0.95	128	20	0.001	40
n225_close Total		0.4166666667	0.08333333333	0.95	74.66666667	30	0.001	56.66666667
sp500_close	2010-10-01	0.5	0.5	0.95	64	20	0.001	20
	2011-10-01	0.5	0.5	0.95	64	20	0.001	50
	2012-10-01	0.5	0.5	0.95	128	50	0.001	40
	2013-10-01	0.5	0	0.95	64	50	0.001	110
	2014-10-01	0.5	0.5	0.95	64	20	0.001	80
	2015-10-01	0.5	0.5	0.95	64	20	0.001	70
sp500_close Total		0.5	0.4166666667	0.95	74.66666667	30	0.001	61.66666667

(a) `basic_rnn_cell`

		model	Values						
		basic_lstm_cell							
target_column	start_date_test	dropout	gradient_norm_threshold	decay_rate	hidden_size	sequence_length	learning_rate	epochs	forget_bias
hsi_close	2010-10-01	0.5	1	0.95	128	50	0.001	70	1
	2011-10-01	0.5	0.5	0.95	128	50	0.001	90	1
	2012-10-01	0	1	0.95	64	50	0.001	110	1
	2013-10-01	0	1	0.95	128	50	0.001	130	1
	2014-10-01	0	0.5	0.95	64	50	0.001	140	1
	2015-10-01	0.5	1	0.95	128	50	0.001	100	1
hsi_close Total		0.25	0.8333333333	0.95	106.6666667	50	0.001	106.6666667	1
n225_close	2010-10-01	0.5	0.5	0.95	128	20	0.001	70	1
	2011-10-01	0.5	0.5	0.95	128	20	0.001	80	1
	2012-10-01	0	0.5	0.95	128	20	0.001	40	1
	2013-10-01	0.5	0.5	0.95	128	20	0.001	90	1
	2014-10-01	0.5	0.5	0.95	128	50	0.001	80	1
	2015-10-01	0	1	0.95	128	50	0.001	120	1
n225_close Total		0.3333333333	0.5833333333	0.95	128	30	0.001	80	1
sp500_close	2010-10-01	0.5	0.5	0.95	128	50	0.001	100	1
	2011-10-01	0.5	1	0.95	128	50	0.001	130	1
	2012-10-01	0.5	1	0.95	128	50	0.001	120	1
	2013-10-01	0	0.5	0.95	64	50	0.001	120	1
	2014-10-01	0.5	0.5	0.95	128	50	0.001	80	1
	2015-10-01	0.5	0.5	0.95	128	50	0.001	110	1
sp500_close Total		0.4166666667	0.6666666667	0.95	117.3333333	50	0.001	110	1

(b) `basic_lstm_cell`

Figure 16: Hyperparameters averaged and summarised across model, data set and time frame.

		model	Values					
		gru_cell						
target_column	start_date_test	dropout	gradient_norm_threshold	decay_rate	hidden_size	sequence_length	learning_rate	epochs
hsi_close	2010-10-01	0.5	1	0.95	64	50	0.001	110
	2011-10-01	0.5	1	0.95	128	50	0.001	90
	2012-10-01	0	1	0.95	128	50	0.001	100
	2013-10-01	0	1	0.95	128	50	0.001	80
	2014-10-01	0.5	0.5	0.95	128	50	0.001	80
	2015-10-01	0.5	1	0.95	128	50	0.001	80
hsi_close Total		0.3333333333	0.9166666667	0.95	117.3333333	50	0.001	90
n225_close	2010-10-01	0.5	1	0.95	128	50	0.001	90
	2011-10-01	0	0.5	0.95	64	50	0.001	90
	2012-10-01	0	0.5	0.95	128	50	0.001	110
	2013-10-01	0	1	0.95	128	50	0.001	90
	2014-10-01	0	0.5	0.95	128	50	0.001	70
	2015-10-01	0	1	0.95	128	50	0.001	120
n225_close Total		0.08333333333	0.75	0.95	117.3333333	50	0.001	95
sp500_close	2010-10-01	0.5	1	0.95	128	50	0.001	100
	2011-10-01	0.5	0.5	0.95	128	50	0.001	80
	2012-10-01	0	1	0.95	128	50	0.001	100
	2013-10-01	0.5	1	0.95	128	50	0.001	120
	2014-10-01	0.5	0.5	0.95	128	50	0.001	80
	2015-10-01	0.5	0.5	0.95	64	50	0.001	120
sp500_close Total		0.4166666667	0.75	0.95	117.3333333	50	0.001	100

(a) `gru_cell`

		model	Values						
		peep_lstm_cell							
target_column	start_date_test	dropout	gradient_norm_threshold	decay_rate	hidden_size	sequence_length	learning_rate	epochs	forget_bias
hsi_close	2010-10-01	0	0	0.95	128	20	0.001	40	1
	2011-10-01	0.5	0.5	0.95	128	20	0.001	70	1
	2012-10-01	0	0	0.95	64	50	0.001	100	1
	2013-10-01	0	0	0.95	64	20	0.001	100	1
	2014-10-01	0	0.5	0.95	64	20	0.001	30	1
	2015-10-01	0.5	0.5	0.95	128	20	0.001	130	1
hsi_close Total		0.1666666667	0.25	0.95	96	25	0.001	78.33333333	1
n225_close	2010-10-01	0	0.5	0.95	64	20	0.001	40	1
	2011-10-01	0	0	0.95	64	20	0.001	130	1
	2012-10-01	0	0.5	0.95	64	20	0.001	70	1
	2013-10-01	0.5	0.5	0.95	128	20	0.001	70	1
	2014-10-01	0.5	0.5	0.95	128	20	0.001	40	1
	2015-10-01	0.5	0.5	0.95	128	50	0.001	90	1
n225_close Total		0.25	0.4166666667	0.95	96	25	0.001	73.33333333	1
sp500_close	2010-10-01	0.5	0.5	0.95	128	20	0.001	60	1
	2011-10-01	0.5	0	0.95	128	50	0.001	180	1
	2012-10-01	0.5	0.5	0.95	128	20	0.001	80	1
	2013-10-01	0.5	0.5	0.95	128	20	0.001	130	1
	2014-10-01	0.5	0.5	0.95	128	50	0.001	90	1
	2015-10-01	0	0	0.95	128	20	0.001	80	1
sp500_close Total		0.4166666667	0.3333333333	0.95	128	30	0.001	103.3333333	1

(b) `peep_lstm_cell`

Figure 17: Hyperparameters averaged and summarised across model, data set and time frame.

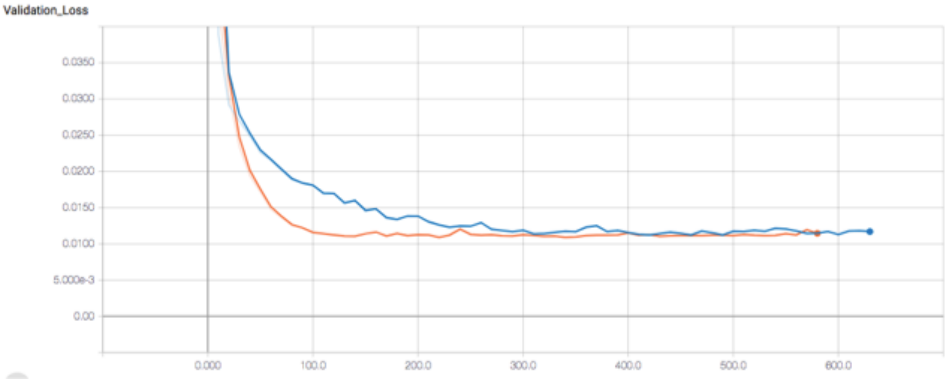
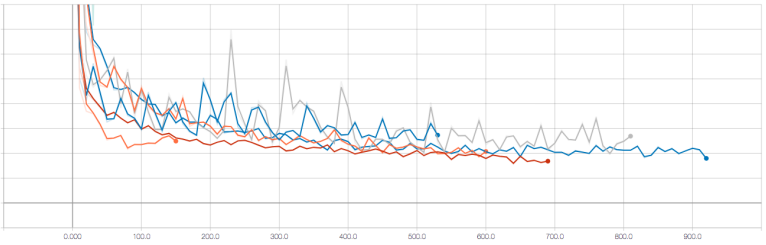
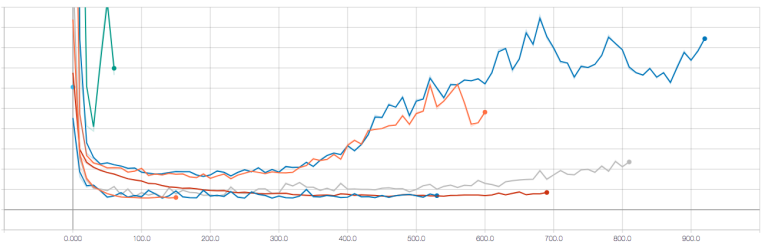


Figure 18: Validation losses for sequences with optimal hyperparameters with (orange, $T = 50$) and without truncation (T = approximately 500 trading days, blue) of sequences respectively.



(a) Training error



(b) Validation error

Figure 19: Training and validation errors for models with combinations of 2 RNN layers and 256 or 512 RNN blocks. As can be seen, generalization performance suffers.

6.1.2.10 Preserving states As noted in Section 5.5.2.1, we implement our RNN networks to be able to preserve hidden states between truncated sequences both during training and between training and the final out-of-sample prediction. Our empirical findings suggest, that *not* preserving state between training sequences while preserving between validation sequences yields the best generalization performance. In all cases do we pass the resulting state of the training pass as initial state of the final out-of-sample prediction.

6.1.3 Existing literature benchmarks

Table 20 presents results with benchmark measures from the existing literature, where

- `LSTM_plos` the baseline LSTM model of [5]
- `WSAEs-LSTM` the proposed model of [5]

Further, results from the `gru_cell` are compared against both the `LSTM_plos` and `WSAEs-LSTM` by itself in Table 22. As can be seen, the `WSAEs-LSTM` performs better than the `basic_lstm_cell` and the three other RNN implementations of this thesis in only 1 of 18 cases. Adding to this; to the best of our knowledge, the reported results come from performing a single run and thus do not take into account the possibility, that the random initialization could have favored one particular configuration over another[15].

For comparison, Figure 21 presents the max R measure obtained across all runs. In this case none of the out-of-sample test periods fall out in favor of the `WSAEs-LSTM` when compared to the RNN architectures and we conclude the comparison with the following remarks:

- Although the data set does not allow us to conclude so with statistical significance; when compared to the `gru_cell`, the advanced preprocessing setup of the `WSAEs-LSTM` does not perform better.
- In all cases did the `gru_cell` implementation perform superior to the baseline `LSTM_plos` implementation by which the proposed model was compared. No details on implementation or final architecture are enclosed.
- In all cases did `t-1` perform superior to the proposed model.
- Allowing random initialization and configurations to determine outcome does have an impact.

[15] We have tried getting in touch with the authors for double confirmation of their approach, but no response has been made.

MEDIAN of r		model					
target_column	start_date_test	basic_lstm_cell	basic_rnn_cell	gru_cell	LSTM_plos	peep_lstm_cell	WSAEs-LSTM
hsi_close	2010-10-01	0.9798	0.9783	0.9794	0.8950	0.9800	0.9440
	2011-10-01	0.9656	0.9652	0.9661	0.8170	0.9653	0.9240
	2012-10-01	0.9666	0.9641	0.9673	0.7270	0.9642	0.9200
	2013-10-01	0.9727	0.9758	0.9724	0.8120	0.9761	0.9270
	2014-10-01	0.9839	0.9828	0.9866	0.9320	0.9858	0.9040
	2015-10-01	0.9817	0.9759	0.9812	0.9010	0.9787	0.9680
hsi_close Total		0.9749	0.9755	0.9752	0.8560	0.9761	0.9255
n225_close	2010-10-01	0.9719	0.9698	0.9715	0.7590	0.9719	0.8950
	2011-10-01	0.9783	0.9719	0.9791	0.7590	0.9573	0.9270
	2012-10-01	0.9917	0.8754	0.9832	0.9720	0.9511	0.9920
	2013-10-01	0.9393	0.8905	0.9469	0.5960	0.9522	0.8850
	2014-10-01	0.9714	0.9570	0.9771	0.9180	0.9816	0.9740
	2015-10-01	0.9743	0.9719	0.9745	0.8810	0.9742	0.9510
n225_close Total		0.9742	0.9662	0.9744	0.8200	0.9708	0.9390
sp500_close	2010-10-01	0.9728	0.8944	0.9699	0.8730	0.9710	0.9440
	2011-10-01	0.9821	0.9786	0.9808	0.9050	0.9805	0.9440
	2012-10-01	0.9898	0.9683	0.9888	0.9680	0.9935	0.9840
	2013-10-01	0.9596	0.8462	0.9758	0.9530	0.9743	0.9730
	2014-10-01	0.9173	0.9405	0.9305	0.7950	0.9489	0.8800
	2015-10-01	0.9746	0.9743	0.9717	0.7750	0.9760	0.9530
sp500_close Total		0.9732	0.9495	0.9744	0.8890	0.9760	0.9485

Figure 20: Benchmark with existing literature by median of R measure.

MAX of r		model					
target_column	start_date_test	basic_lstm_cell	basic_rnn_cell	gru_cell	LSTM_plos	peep_lstm_cell	WSAEs-LSTM
hsi_close	2010-10-01	0.9807	0.9815	0.9812	0.8950	0.9802	0.9440
	2011-10-01	0.9663	0.9658	0.9665	0.8170	0.9659	0.9240
	2012-10-01	0.9667	0.9677	0.9676	0.7270	0.9656	0.9200
	2013-10-01	0.9742	0.9764	0.9741	0.8120	0.9767	0.9270
	2014-10-01	0.9852	0.9847	0.9868	0.9320	0.9867	0.9040
	2015-10-01	0.9818	0.9820	0.9814	0.9010	0.9798	0.9680
hsi_close Total		0.9852	0.9847	0.9868	0.9320	0.9867	0.9680
n225_close	2010-10-01	0.9723	0.9712	0.9716	0.7590	0.9722	0.8950
	2011-10-01	0.9792	0.9735	0.9793	0.7590	0.9766	0.9270
	2012-10-01	0.9935	0.9006	0.9896	0.9720	0.9703	0.9920
	2013-10-01	0.9588	0.9584	0.9598	0.5960	0.9558	0.8850
	2014-10-01	0.9833	0.9805	0.9843	0.9180	0.9866	0.9740
	2015-10-01	0.9745	0.9736	0.9746	0.8810	0.9743	0.9510
n225_close Total		0.9935	0.9805	0.9896	0.9720	0.9866	0.9920
sp500_close	2010-10-01	0.9731	0.9474	0.9722	0.8730	0.9730	0.9440
	2011-10-01	0.9821	0.9817	0.9818	0.9050	0.9815	0.9440
	2012-10-01	0.9933	0.9859	0.9917	0.9680	0.9938	0.9840
	2013-10-01	0.9871	0.9464	0.9901	0.9530	0.9863	0.9730
	2014-10-01	0.9516	0.9466	0.9534	0.7950	0.9534	0.8800
	2015-10-01	0.9759	0.9752	0.9761	0.7750	0.9763	0.9530
sp500_close Total		0.9933	0.9859	0.9917	0.9680	0.9938	0.9840

Figure 21: Benchmark with existing literature by max of R measure.

MEDIAN of r		model	
target_column	start_date_test	gru_cell	WSAEs-LSTM
hsi_close	2010-10-01	0.9794	0.9440
	2011-10-01	0.9661	0.9240
	2012-10-01	0.9673	0.9200
	2013-10-01	0.9724	0.9270
	2014-10-01	0.9866	0.9040
	2015-10-01	0.9812	0.9680
hsi_close Total		0.9752	0.9255
n225_close	2010-10-01	0.9715	0.8950
	2011-10-01	0.9791	0.9270
	2012-10-01	0.9832	0.9920
	2013-10-01	0.9469	0.8850
	2014-10-01	0.9771	0.9740
	2015-10-01	0.9745	0.9510
n225_close Total		0.9744	0.9390
sp500_close	2010-10-01	0.9692	0.9440
	2011-10-01	0.9808	0.9440
	2012-10-01	0.9914	0.9840
	2013-10-01	0.9893	0.9730
	2014-10-01	0.9477	0.8800
	2015-10-01	0.9755	0.9530
sp500_close Total		0.9760	0.9485

(a) `gru_cell/LSTM_plos`

MEDIAN of r		model	
target_column	start_date_test	gru_cell	LSTM_plos
hsi_close	2010-10-01	0.9794	0.8950
	2011-10-01	0.9661	0.8170
	2012-10-01	0.9673	0.7270
	2013-10-01	0.9724	0.8120
	2014-10-01	0.9866	0.9320
	2015-10-01	0.9812	0.9010
hsi_close Total		0.9752	0.8560
n225_close	2010-10-01	0.9715	0.7590
	2011-10-01	0.9791	0.7590
	2012-10-01	0.9832	0.9720
	2013-10-01	0.9469	0.5960
	2014-10-01	0.9771	0.9180
	2015-10-01	0.9745	0.8810
n225_close Total		0.9744	0.8200
sp500_close	2010-10-01	0.9692	0.8730
	2011-10-01	0.9808	0.9050
	2012-10-01	0.9914	0.9680
	2013-10-01	0.9893	0.9530
	2014-10-01	0.9477	0.7950
	2015-10-01	0.9755	0.7750
sp500_close Total		0.9760	0.8890

(b) `gru_cell/WSAEs-LSTM`

MEDIAN of r		model	
target_column	start_date_test	gru_cell	t-1
hsi_close	2010-10-01	0.9794	0.9829
	2011-10-01	0.9661	0.9659
	2012-10-01	0.9673	0.9682
	2013-10-01	0.9724	0.9768
	2014-10-01	0.9866	0.9870
	2015-10-01	0.9812	0.9821
hsi_close Total		0.9752	0.9794
n225_close	2010-10-01	0.9715	0.9716
	2011-10-01	0.9791	0.9801
	2012-10-01	0.9832	0.9944
	2013-10-01	0.9469	0.9589
	2014-10-01	0.9771	0.9876
	2015-10-01	0.9745	0.9742
n225_close Total		0.9744	0.9772
sp500_close	2010-10-01	0.9692	0.9724
	2011-10-01	0.9808	0.9820
	2012-10-01	0.9914	0.9937
	2013-10-01	0.9893	0.9900
	2014-10-01	0.9477	0.9542
	2015-10-01	0.9755	0.9762
sp500_close Total		0.9760	0.9791

(c) `gru_cell/t-1`

MEDIAN of r		model	
target_column	start_date_test	t-1	WSAEs-LSTM
hsi_close	2010-10-01	0.9829	0.9440
	2011-10-01	0.9659	0.9240
	2012-10-01	0.9682	0.9200
	2013-10-01	0.9768	0.9270
	2014-10-01	0.9870	0.9040
	2015-10-01	0.9821	0.9680
hsi_close Total		0.9794	0.9255
n225_close	2010-10-01	0.9716	0.8950
	2011-10-01	0.9801	0.9270
	2012-10-01	0.9944	0.9920
	2013-10-01	0.9589	0.8850
	2014-10-01	0.9876	0.9740
	2015-10-01	0.9742	0.9510
n225_close Total		0.9772	0.9390
sp500_close	2010-10-01	0.9724	0.9440
	2011-10-01	0.9820	0.9440
	2012-10-01	0.9937	0.9840
	2013-10-01	0.9900	0.9730
	2014-10-01	0.9542	0.8800
	2015-10-01	0.9762	0.9530
sp500_close Total		0.9791	0.9485

(d) `WSAEs-LSTM/t-1`

Figure 22: Benchmark with existing literature by median of R measure.

6.2 Benchmarking

Finally, in Table 23 we presents the results of the stochastic models of the RiskButler portfolio, where:

- `GBM` is the stochastic process described in Section 3.1.3.1
- `CKLS 1992` is the stochastic process described in Section 3.1.3.2

We limit focus to `gru_cell` in the interest of clarity as this was concluded to perform the best across various parameters in Section 6.1.1.

6.2.1 Stochastic models

As can be seen; in all but one case for the `GBM` and two cases for the `CKLS 1992` did the stochastic models of the RiskButler portfolio outperform the `gru_cell`. While this does come a little surprisingly, given the advances with deep learning neural networks in other domains, it does confirm the time proven, common knowledge established in the financial industry.

Limited access to actual predictions and results made it not possible to assess and compare statistical properties.

Section 7 explores some possible explanations and tests some concrete mitigation attempts before concluding with thoughts and suggestions for further explorations in the field.

7 Discussion

We start the section by reiterating two important considerations: First, neural networks are notoriously difficult to train, and we may not have uncovered the optimal hyperparameter configuration in this thesis.

And second, we followed the prediction approach described in [5] closely. Most importantly, this meant 1) limiting training data set to a 2 year period and 2) performing predictions on an entire years worth of trading days.

7.1 Resilient Backpropagation optimization algorithms

To address the first concern, an ideal solution would be to eliminate the need for the hyperparameter tuning phase as described in Section 5.

During the writing of this thesis, a TensorFlow implementation of the Resilient Backpropagation (Rprop) algorithms [41] and their Improved Resilient Backpropagation (iRprop) successors [23, 24] became available[16], all

[16] At the time of writing, a pre-compiled binary of the TensorFlow library is needed for conducting experiments and reproducing results with the Rprop algorithms. Contact the author for more information.

MEDIAN of r2		model	
target_column	start_date_test	GBM	gru_cell
hsi_close	2010-10-01	0.9678	0.9592
	2011-10-01	0.9345	0.9333
	2012-10-01	0.9424	0.9356
	2013-10-01	0.9543	0.9455
	2014-10-01	0.9752	0.9734
	2015-10-01	0.9657	0.9628
hsi_close Total		0.9600	0.9510
n225_close	2010-10-01	0.9244	0.9438
	2011-10-01	0.9616	0.9586
	2012-10-01	0.9894	0.9667
	2013-10-01	0.9244	0.8965
	2014-10-01	0.9757	0.9547
	2015-10-01	0.9514	0.9496
n225_close Total		0.9565	0.9494
sp500_close	2010-10-01	0.9471	0.9393
	2011-10-01	0.9666	0.9620
	2012-10-01	0.9882	0.9828
	2013-10-01	0.9806	0.9787
	2014-10-01	0.9129	0.8981
	2015-10-01	0.9553	0.9516
sp500_close Total		0.9610	0.9525

(a) `gru_cell/GBM`

MEDIAN of r2		model	
target_column	start_date_test	CKLS 1992	gru_cell
hsi_close	2010-10-01	0.9678	0.9592
	2011-10-01	0.0102	0.9333
	2012-10-01	0.9426	0.9356
	2013-10-01	0.9543	0.9455
	2014-10-01	0.9752	0.9734
	2015-10-01	0.9656	0.9628
hsi_close Total		0.9600	0.9510
n225_close	2010-10-01	0.9243	0.9438
	2011-10-01	0.9616	0.9586
	2012-10-01	0.9893	0.9667
	2013-10-01	0.9243	0.8965
	2014-10-01	0.9757	0.9547
	2015-10-01	0.9515	0.9496
n225_close Total		0.9566	0.9494
sp500_close	2010-10-01	0.9806	0.9393
	2011-10-01	0.9666	0.9620
	2012-10-01	0.9882	0.9828
	2013-10-01	0.9806	0.9787
	2014-10-01	0.9129	0.8981
	2015-10-01	0.9553	0.9516
sp500_close Total		0.9736	0.9525

(b) `gru_cell/CKLS 1992`

MEDIAN of r2		model	
target_column	start_date_test	GBM	t-1
hsi_close	2010-10-01	0.9678	0.9660
	2011-10-01	0.9345	0.9329
	2012-10-01	0.9424	0.9374
	2013-10-01	0.9543	0.9542
	2014-10-01	0.9752	0.9742
	2015-10-01	0.9657	0.9644
hsi_close Total		0.9600	0.9593
n225_close	2010-10-01	0.9244	0.9440
	2011-10-01	0.9616	0.9606
	2012-10-01	0.9894	0.9889
	2013-10-01	0.9244	0.9194
	2014-10-01	0.9757	0.9754
	2015-10-01	0.9514	0.9491
n225_close Total		0.9565	0.9548
sp500_close	2010-10-01	0.9471	0.9456
	2011-10-01	0.9666	0.9644
	2012-10-01	0.9882	0.9875
	2013-10-01	0.9806	0.9800
	2014-10-01	0.9129	0.9106
	2015-10-01	0.9553	0.9529
sp500_close Total		0.9610	0.9587

(c) `GBM/t-1`

MEDIAN of r2		model	
target_column	start_date_test	CKLS 1992	t-1
hsi_close	2010-10-01	0.9678	0.9660
	2011-10-01	0.0102	0.9329
	2012-10-01	0.9426	0.9374
	2013-10-01	0.9543	0.9542
	2014-10-01	0.9752	0.9742
	2015-10-01	0.9656	0.9644
hsi_close Total		0.9600	0.9593
n225_close	2010-10-01	0.9243	0.9440
	2011-10-01	0.9616	0.9606
	2012-10-01	0.9893	0.9889
	2013-10-01	0.9243	0.9194
	2014-10-01	0.9757	0.9754
	2015-10-01	0.9515	0.9491
n225_close Total		0.9566	0.9548
sp500_close	2010-10-01	0.9806	0.9456
	2011-10-01	0.9666	0.9644
	2012-10-01	0.9882	0.9875
	2013-10-01	0.9806	0.9800
	2014-10-01	0.9129	0.9106
	2015-10-01	0.9553	0.9529
sp500_close Total		0.9736	0.9587

(d) `CKLS 1992/t-1`

Figure 23: `gru_cell` benchmark with stochastic models by median R^2 measure

	$\texttt{adam}_{\{dataset\}}$
$\texttt{i\_rprop\_plus}_{sp500}$	0.4289633889
$\texttt{i\_rprop\_plus}_{hsi}$	0.3952670113
$\texttt{i\_rprop\_plus}_{n225}$	0.3710770321
$\texttt{i\_rprop\_minus}_{sp500}$	0.2972716891
$\texttt{i\_rprop\_minus}_{hsi}$	0.3952670113
$\texttt{i\_rprop\_minus}_{n225}$	0.5692201637
$\texttt{rprop\_plus}_{sp500}$	0.3255265868
$\texttt{rprop\_plus}_{hsi}$	0.6520436224
$\texttt{rprop\_plus}_{n225}$	0.2339889163
$\texttt{rprop\_minus}_{sp500}$	0.3710770321
$\texttt{rprop\_minus}_{hsi}$	0.2170168245
$\texttt{rprop\_minus}_{n225}$	0.5592305358

Table 5: p values of two-sided Mann-Whitney U test comparing Rprop algorithms to Adam optimization setup, grouped by data set.

of which – in addition to being fast and accurate – promise a robustness with respect to the internal parameters, and insensitivity to network topology that make them well suited for addressing exactly this.

An in depth explanation of the dynamics is beyond the scope of this thesis. Instead the reader is referred to the original articles above.

Table 24 presents results from the four Rprop algorithms grouped by data set and time frame compared to the tuned Adam optimizer used throughout this thesis.

- `adam` the GRU cell results of Table 10a
- `i_rprop_minus` the iRprop$^-$ optimization algorithm
- `i_rprop_plus` the iRprop$^+$ optimization algorithm
- `rprop_minus` the Rprop$^-$ optimization algorithm
- `rprop_plus` the Rprop$^+$ optimization algorithm

With just default configuration, drop out regularization at 0.5, the GRU architecture and otherwise following the prediction approach described in Section 5, the results are remarkably close to results from the elaborate hyperparameter tuning scheme for the Adam optimizer. In fact, as shown in Table 5; at a significance level of 0.05 we cannot tell the finely tuned Adam algorithm from the Rprop methods with default configurations. However, we still cannot outperform the stochastic models of the RiskButler portfolio.

MEDIAN of r2		optimizer				
target_column	start_date_test	adam	i_rprop_minus	i_rprop_plus	rprop_minus	rprop_plus
hsi_close	2010-10-01	0.9592	0.9618	0.9621	0.9597	0.9615
	2011-10-01	0.9333	0.9303	0.9275	0.9296	0.9305
	2012-10-01	0.9356	0.9296	0.9317	0.9290	0.9319
	2013-10-01	0.9455	0.9423	0.9455	0.9415	0.9425
	2014-10-01	0.9734	0.9696	0.9687	0.9694	0.9714
	2015-10-01	0.9628	0.9625	0.9638	0.9606	0.9631
hsi_close Total		0.9510	0.9518	0.9534	0.9508	0.9523
n225_close	2010-10-01	0.9438	0.9312	0.9352	0.9406	0.9302
	2011-10-01	0.9586	0.9536	0.9542	0.9526	0.9521
	2012-10-01	0.9667	0.9864	0.9874	0.9883	0.9871
	2013-10-01	0.8965	0.8717	0.8760	0.8916	0.8962
	2014-10-01	0.9547	0.9570	0.9448	0.9454	0.9313
	2015-10-01	0.9496	0.9457	0.9500	0.9487	0.9442
n225_close Total		0.9494	0.9457	0.9497	0.9474	0.9428
sp500_close	2010-10-01	0.9393	0.9401	0.9422	0.9402	0.9432
	2011-10-01	0.9620	0.9625	0.9628	0.9613	0.9624
	2012-10-01	0.9828	0.9789	0.9760	0.9765	0.9778
	2013-10-01	0.9787	0.9460	0.9541	0.9531	0.9424
	2014-10-01	0.8981	0.8608	0.8723	0.8649	0.7336
	2015-10-01	0.9516	0.9394	0.9417	0.9414	0.9404
sp500_close Total		0.9525	0.9432	0.9452	0.9423	0.9433

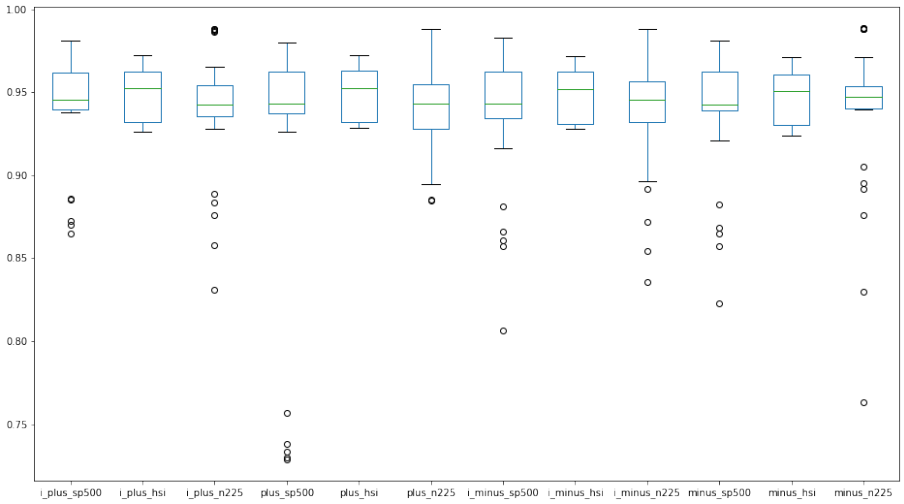
Figure 24: Comparison between the default Rprop and elaborate hyperparameter tuning scheme combined with the Adam optimizations algorithms.

MEDIAN of r2		optimizer	
target_column	start_date_test	adam	i_rprop_plus
hsi_close	2010-10-01	0.9592	0.9621
	2011-10-01	0.9333	0.9275
	2012-10-01	0.9356	0.9317
	2013-10-01	0.9455	0.9455
	2014-10-01	0.9734	0.9687
	2015-10-01	0.9628	0.9638
hsi_close Total		0.9510	0.9534
n225_close	2010-10-01	0.9438	0.9352
	2011-10-01	0.9586	0.9542
	2012-10-01	0.9667	0.9874
	2013-10-01	0.8965	0.8760
	2014-10-01	0.9547	0.9448
	2015-10-01	0.9496	0.9500
n225_close Total		0.9494	0.9497
sp500_close	2010-10-01	0.9393	0.9422
	2011-10-01	0.9620	0.9628
	2012-10-01	0.9828	0.9760
	2013-10-01	0.9787	0.9541
	2014-10-01	0.8981	0.8723
	2015-10-01	0.9516	0.9417
sp500_close Total		0.9525	0.9452

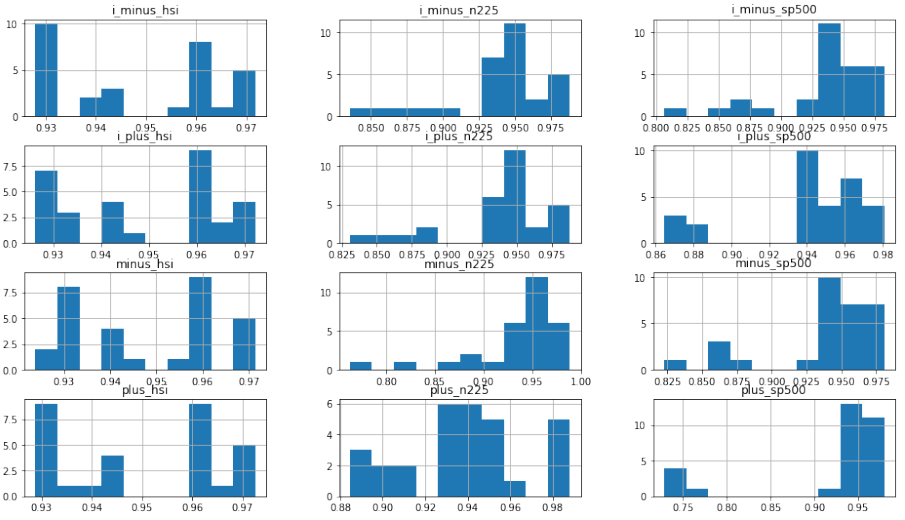
Figure 25: Median R^2 comparison between tuned Adam optimizations algorithm and the iRprop$^+$ method.

	i_plus_sp500	i_plus_hsi	i_plus_n225	plus_sp500	plus_hsi	plus_n225	i_minus_sp500	i_minus_hsi	i_minus_n225	minus_sp500	minus_hsi	minus_n225
count	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000
mean	0.941483	0.949123	0.939221	0.917448	0.950001	0.939389	0.935736	0.949260	0.939920	0.938512	0.948050	0.937362
std	0.032671	0.016298	0.038039	0.083290	0.016390	0.030275	0.042020	0.016354	0.037571	0.039847	0.016226	0.048067
min	0.864568	0.926309	0.830832	0.728873	0.928796	0.884572	0.806663	0.927958	0.835456	0.822907	0.923803	0.762913
25%	0.939799	0.931817	0.935398	0.937238	0.932132	0.927784	0.934473	0.930761	0.931795	0.938884	0.930233	0.940252
50%	0.945187	0.952214	0.942421	0.943305	0.952299	0.942833	0.943231	0.951792	0.945693	0.942345	0.950816	0.947448
75%	0.961910	0.962589	0.954244	0.962396	0.963022	0.954606	0.962687	0.962515	0.956492	0.962505	0.960538	0.953361
max	0.980900	0.972322	0.987962	0.979894	0.972586	0.988100	0.982642	0.971805	0.987992	0.980830	0.971444	0.988762

(a) Statistical summaries grouped by model and data set



(b) Box plots grouped by model and data set.



(c) Histograms grouped by model and data set.

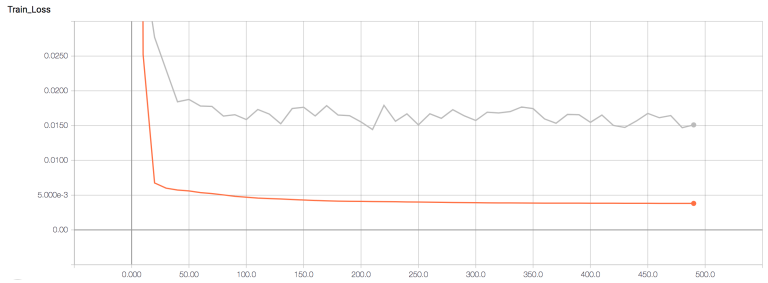
MEDIAN of r2		model	
target_column	start_date_test	gru_cell	gru_cell_5y
hsi_close	2010-10-01	0.9592	0.9653
	2011-10-01	0.9333	0.9345
	2012-10-01	0.9356	0.9383
	2013-10-01	0.9455	0.9427
	2014-10-01	0.9734	0.9779
	2015-10-01	0.9628	0.9687
hsi_close Total		0.9510	0.9586
n225_close	2010-10-01	0.9438	0.9371
	2011-10-01	0.9586	0.9645
	2012-10-01	0.9667	0.9885
	2013-10-01	0.8965	0.8595
	2014-10-01	0.9547	0.9677
	2015-10-01	0.9496	0.9508
n225_close Total		0.9494	0.9518
sp500_close	2010-10-01	0.9408	0.9419
	2011-10-01	0.9622	0.9610
	2012-10-01	0.9778	0.9843
	2013-10-01	0.9521	0.9356
	2014-10-01	0.8657	0.8360
	2015-10-01	0.9442	0.9367
sp500_close Total		0.9494	0.9428

Figure 27: R^2 comparison between otherwise optimal hyperparameter configuration and the GRU architecture with 2 and 5 years worth of training samples respectively.

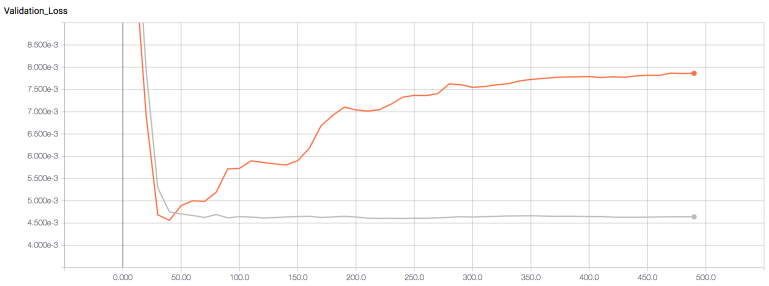
7.2 Expanding the data set

To address 1) of the second concern, adding more training samples may allow for a more complex model, where performance does not saturate as quickly as we saw in Section 6.1.2.8. And with truncation of sequences combined with the properties of the RNNs and the regularization scheme defined in Section 5.5, we would expect this to at least give us the same results. Figure 27 presents results from a 5 year training period in comparison with the 2 year period used throughout this thesis. As can be seen, predictive performance increases in some cases but is decreased in others.

An interesting observation though, is presented in Figure 28: while applying dropout did not always improve performance on the 2 year training data set, in all cases did we see gains with 5 years' worth of data. This presented case is in the extreme, however, it illustrates well how the training error is worsened while the validation error is improved. Otherwise optimal hyperparameters.



(a) Training error



(b) Validation error

Figure 28: Training and validation error for 5 years training data with (grey) and without (orange) drop out at 0.5 and otherwise optimal hyperparameter configuration.

MEDIAN of r2		model	
target_column	start_date_test	gru_cell	gru_cell*
hsi_close	2010-10-01	0.9592	0.9632
	2011-10-01	0.9333	0.9310
	2012-10-01	0.9356	0.9340
	2013-10-01	0.9455	0.9515
	2014-10-01	0.9734	0.9730
	2015-10-01	0.9628	0.9622
hsi_close Total		0.9510	0.9569
n225_close	2010-10-01	0.9438	0.9440
	2011-10-01	0.9586	0.9566
	2012-10-01	0.9667	0.9881
	2013-10-01	0.8965	0.8959
	2014-10-01	0.9547	0.9677
	2015-10-01	0.9496	0.9480
n225_close Total		0.9494	0.9523
sp500_close	2010-10-01	0.9408	0.9426
	2011-10-01	0.9622	0.9627
	2012-10-01	0.9778	0.9851
	2013-10-01	0.9521	0.9746
	2014-10-01	0.8657	0.8840
	2015-10-01	0.9442	0.9529
sp500_close Total		0.9494	0.9578

(a) `gru_cell/gru_cell*`

MEDIAN of r2		model	
target_column	start_date_test	GBM	gru_cell*
hsi_close	2010-10-01	0.9680	0.9632
	2011-10-01	0.9350	0.9310
	2012-10-01	0.9420	0.9340
	2013-10-01	0.9540	0.9515
	2014-10-01	0.9750	0.9730
	2015-10-01	0.9660	0.9622
hsi_close Total		0.9600	0.9569
n225_close	2010-10-01	0.9240	0.9440
	2011-10-01	0.9620	0.9566
	2012-10-01	0.9890	0.9881
	2013-10-01	0.9240	0.8959
	2014-10-01	0.9760	0.9677
	2015-10-01	0.9510	0.9480
n225_close Total		0.9565	0.9523
sp500_close	2010-10-01	0.9470	0.9426
	2011-10-01	0.9670	0.9627
	2012-10-01	0.9880	0.9851
	2013-10-01	0.9810	0.9746
	2014-10-01	0.9130	0.8840
	2015-10-01	0.9550	0.9529
sp500_close Total		0.9610	0.9578

(b) `gru_cell*/GBM`

Figure 29: `gru_cell` model comparison by R^2 measure with and without rolling training pass and against the `GBM` model. Asterisk mark rolling.

7.3 Rolling time horizon

To address 2) of the second concern, we compare results from performing a single training pass to results with a rolling time horizon. The latter updates the training data after every out-of-sample prediction and thus incorporates the last observed information in the model.

Table 29 presents the predictions from a single and a rolling training pass of the `gru_cell` respectively. While predictive accuracy with a rolling training pass does outperform predictions from a single training pass, it still does not change anything when compared to the stochastic models.

7.4 Expanding the network architecture

The proposed model of [5] derives its strength from an advanced preprocessing setup of stacked auto encoders to *"learn the deep features of financial time series in an unsupervised manner"*. An in depth understanding and implementation is beyond the scope of this thesis. Two modifications are tried: 1) adding simple preprocessing step consisting of a single feed forward layer before passing the input to the RNN layer and 2) adding a feed forward layer after passing the input through the RNN layer. Both of these modifications might allow the model to project the sequences into a space with easier temporal dynamics which could improve predictive performance.

Figure 30 presents results from the iRprop$^+$ optimizer on a 5 year training period with a 256 unit feed forward layer before the RNN layer in com-

MEDIAN of r2		model	
target_column	start_date_test	gru_cell	i_plus_dense
hsi_close	2010-10-01	0.9592	0.9653
	2011-10-01	0.9333	0.9312
	2012-10-01	0.9356	0.9348
	2013-10-01	0.9455	0.9503
	2014-10-01	0.9734	0.9712
	2015-10-01	0.9628	0.9641
hsi_close Total		0.9510	0.9575
n225_close	2010-10-01	0.9438	0.9445
	2011-10-01	0.9586	0.9580
	2012-10-01	0.9667	0.9862
	2013-10-01	0.8965	0.9045
	2014-10-01	0.9547	0.9349
	2015-10-01	0.9496	0.9479
n225_close Total		0.9494	0.9478
sp500_close	2010-10-01	0.9393	0.9447
	2011-10-01	0.9620	0.9624
	2012-10-01	0.9828	0.9853
	2013-10-01	0.9787	0.9616
	2014-10-01	0.8981	0.9015
	2015-10-01	0.9516	0.9473
sp500_close Total		0.9525	0.9485

(a) `i_plus_dense/gru_cell`

MEDIAN of r2		model	
target_column	start_date_test	GBM	i_plus_dense
hsi_close	2010-10-01	0.9680	0.9653
	2011-10-01	0.9350	0.9312
	2012-10-01	0.9420	0.9348
	2013-10-01	0.9540	0.9503
	2014-10-01	0.9750	0.9712
	2015-10-01	0.9660	0.9641
hsi_close Total		0.9600	0.9575
n225_close	2010-10-01	0.9240	0.9445
	2011-10-01	0.9620	0.9580
	2012-10-01	0.9890	0.9862
	2013-10-01	0.9240	0.9045
	2014-10-01	0.9760	0.9349
	2015-10-01	0.9510	0.9479
n225_close Total		0.9565	0.9478
sp500_close	2010-10-01	0.9470	0.9447
	2011-10-01	0.9670	0.9624
	2012-10-01	0.9880	0.9853
	2013-10-01	0.9810	0.9616
	2014-10-01	0.9130	0.9015
	2015-10-01	0.9550	0.9473
sp500_close Total		0.9610	0.9485

(b) `i_plus_dense/GBM`

Figure 30: iRprop$^+$ with a preprocessing layer comparison by R^2 measure against the `gru_cell` and the `GBM` model.

parison with the results from the GRU architecture of Table 10a. As can be seen, we do gain an increase in performance but not at a 0.05 significance level and not sufficiently to compare with the stochastic models. Layers of size 64, 128 and 256 were tried.

7.5 Further work

Numerous additions to both feature space, hyperparameters and architecture of the RNN models that might increase predictive performance of the RNN architectures still remains unexplored.

In this section, we briefly outline promising modifications to the learning problem, optimization setup and feature space unraveled while writing this thesis.

7.5.0.1 Ensemble prediction Combining results from multiple models or from the same model at different training iterations with an appropriate weighing scheme could also improve overall performance.

7.5.0.2 Classification versus regression Posing the learning problem as a classification task could make for a simpler analysis. A simple suggestion would be predicting when the closing price at time t is higher than the opening price:

$$\{p^t_{close} > p^t_{open}\} \tag{7.5.1}$$

Turning the learning problem into a classification task would also make it simpler to develop a trading strategy on top.

7.5.0.3 Adam optimization algorithm optimization For the Adam optimization algorithm, only the learning rate α was considered and tuned. One could consider tuning the exponential decay rates through the β_1 and β_2 parameters.

7.5.0.4 Weight Initialization Various initialization strategies exist. For the experiments performed in this thesis, the TensorFlow default initializer was used for initialization of parameter matrices.

7.5.0.5 Batch normalization In our preprocessing step we normalized input to have zero mean and unit variance before feeding it into the model. [27] propose making this normalization step a part of the model architecture by performing the normalization step before each mini batch during training.

7.5.0.6 Input variables As described in Section 6.1.2.9, increasing the input feature space did not increase predictive performance. However, other input variables might illuminate the problem from other angles. Considered throughout this thesis was:

- Using closing values from other/European indices: Exploiting time zone differences, one might get an indication of overall market trend by for instance looking at closing prices on the European indices when forecasting American closing prices.
- Using intra-day trading data: An other option than adding data from a longer time frame would be adding more data from within the same time frame. Throughout this thesis we relied on daily closing prices only. However, ample data is available and an investigation of hourly or even minutely data points may prove worthy.
- Interpreting overall market trends – for instance, the VIX[17] index is a popular measure of stock market volatility.
- Interpreting textual information from newspapers, social media or other sources – [7] propose stock market prediction from sentiment analysis on Twitter feeds.

[17] https://en.wikipedia.org/wiki/VIX

7.5.0.7 In depth understanding of the financial industry As we cannot compare with the stochastic models of the financial industry, an in depth understanding of these models in particular and the dynamics of the industry as a whole might prove worthy of investigation. One could even consider adding outputs from the models as input features to the RNNs.

7.5.0.8 Derive common patterns Section 6.1.1 suggested that different markets exhibit different characteristics, and that some might be easier to anticipate than others. A straight forward conclusion to this would be deriving common patterns across markets, industries or even single stocks and determining which might prove the best sources for profitability.

7.5.0.9 Investigating other architectures A recent paper [4] conclude that *"the common association between sequence modeling and recurrent networks should be reconsidered, and convolutional networks should be regarded as a natural starting point for sequence modeling tasks"*. As such, applying the convolutional network architecture to financial time series or investigating others might prove a worthy undertaking.

8 Conclusion

This thesis revisits recurrent neural network architectures on financial time series. Based on existing literature, we forecast one-step-ahead closing prices on the S&P500, the Hang Seng and the Nikkei market indices across a 6 year period and in the process develop an elaborate hyperparameter grid search optimization scheme.

In total, predictive accuracy and computational performance of four RNN architectures implemented in the TensorFlow machine learning framework are considered; namely the basic RNN architecture, the basic LSTM architecture with and without peephole connections and networks with Gated Recurrent Units. To achieve a robust conclusion, runs for each prediction period is performed 5 times and models compared for statistical significance.

In our experimental comparison, the basic LSTM architecture performed on par with its peephole counterpart and the GRU architecture compared at a 0.05 significance level. A significant distinction could be made when comparing both the LSTMs and the GRU to the the basic RNN architecture. The GRU was concluded as the architecture of choice for the considered type of application, as results were more stable in terms of lower standard deviation and less overall compute time was required. However, none of the RNN architectures including the GRU performed significantly better than the naive benchmark measure of predicting the value at time $t-1$ at time t (`t-1`).

Predictive accuracy of the GRU is benchmarked against recently published LSTM results [5]. Although the data set does not allow us to conclude so with significance, the elaborate preprocessing scheme did not prove superior[18].

Finally, results are benchmarked against stochastic models applied in the the real world financial sector represented by RiskButler. Across all data sets and time periods, these stochastic models proved superior.

Given the advances in other domains using deep neural networks, neither outperforming `t-1` nor the traditional stochastic models comes somewhat surprising, but does in the case of the latter confirm the accuracy of the time proven time series modelling established in the financial industry.

A cautious conjecture may be, that the relatively poor performance could be a contributing factor to the notably few published results on RNNs applied to financial time series. Concluding that others have not made the coupling between the intriguing prospects of successful financial forecasting and the inherent suitability of RNNs in the domain seems implausible at least.

The benchmarked results are followed up by a discussion section where we 1) compare the Resilient and Improved Resilient Backpropagation optimization algorithms to the tuned Adam optimization algorithm, 2) compare a training phase with a rolling training pass to a training phase with a single training pass, 3) add more training samples, and 4) expand the network architecture by adding additional feed forward layers both before and after the recurrent layers. Neither of the modifications improved performance significantly. However, the Rprop algorithms with default parameters performed as well as the finely tuned Adam optimizer and amongst the four, the Improved Rprop$^+$ method proved superior.

We conclude the thesis with a section on further work and future enhancements, suggesting that unexplored modifications to prediction approach, feature space, network architecture and configuration in combination with a deeper understanding of the financial sector could very well bring state-of-the-art performance on financial time series to the realm of deep neural networks. The reader is encouraged to pick up the gauntlet.

Acknowledgements

A special thanks to Christian Igel (University of Copenhagen) and Sinan Gabel (RiskButler) for supervision and sparring.

[18]The benchmark measure thus did also not perform significantly better than `t-1` but no mentioning of this relatively poor performance is enclosed in the paper.

Appendices

A Current LSTM state-of-the-art

Title	Data set	Input variables	Forecast horizon	Error measures	Citations
[5]	CSI 300, Nifty 50, HSI, N225, S&P500 and DJIA indices	Lagged index data, technical indicator, macro economic variables	1 day	MAPE, R, Theil U	4
[9]	China stock market in Shanghai and Shenzhen	Daily record of high, low, open, close, volume	1 day	Accuracy + random prediction	10
[35]	EUR/USD	Daily historical exchange rates	Three and five day forecasts	Percentage growth of profit	12
[11]	S&P500 and the FOREX EUR/USD	Daily price and minute prices respectively	Daily and minutely	Accuracy	5
[2]	Nikkei 225, 10 companies	Newspaper article + opening prices	Undetermined	Profitability	8
[13]	S&P500 from 1992 until 2015	Normalized one day return	1 day	Random, DNN, LOG	1
[31]	S&P500 1992 to 2015	Closing price	1 day	RMSE, R^2	17
[28]	Google's daily stock data	Percentage return of open, high, low, close and volume	Daily	RMSE	5
[36]	USD/JPY + XAU/USD	Undetermined	Daily	RMSE	10

Table 6: Current state-of-the-art using RNNs for financial forecasting grouped by data set, input variables, forecast horizon and error measure

B LSTM notation

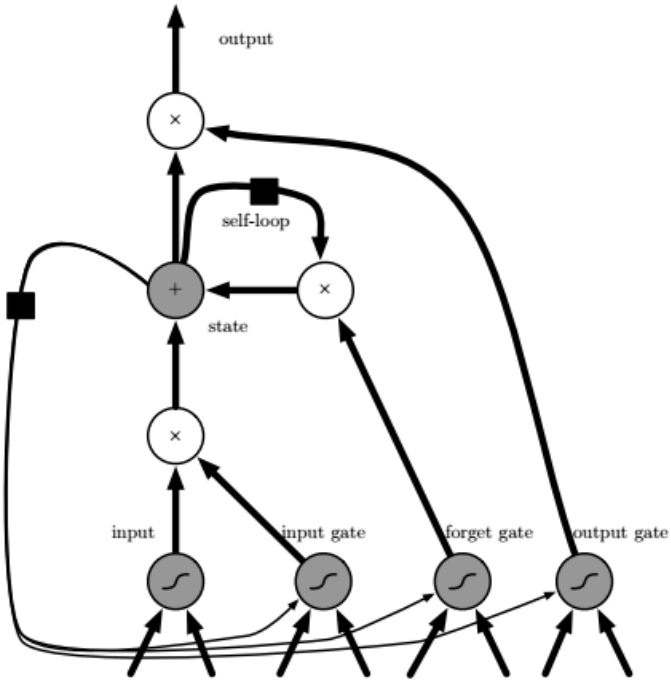


Figure 10.16: Block diagram of the LSTM recurrent network "cell." Cells are connected recurrently to each other, replacing the usual hidden units of ordinary recurrent networks. An input feature is computed with a regular artificial neuron unit. Its value can be accumulated into the state if the sigmoidal input gate allows it. The state unit has a linear self-loop whose weight is controlled by the forget gate. The output of the cell can be shut off by the output gate. All the gating units have a sigmoid nonlinearity, while the input unit can have any squashing nonlinearity. The state unit can also be used as an extra input to the gating units. The black square indicates a delay of a single time step.

Figure 31: `LSTM cell` comprised of a system of gating units and a block as containing multiple cells follwing the notation of [17]

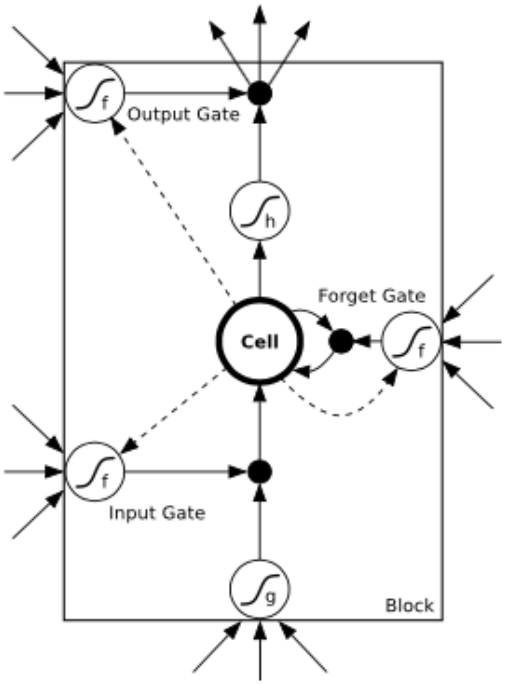


Figure 4.2: **LSTM memory block with one cell.** The three gates are nonlinear summation units that collect activations from inside and outside the block, and control the activation of the cell via multiplications (small black circles). The input and output gates multiply the input and output of the cell while the forget gate multiplies the cell's previous state. No activation function is applied within the cell. The gate activation function 'f' is usually the logistic sigmoid, so that the gate activations are between 0 (gate closed) and 1 (gate open). The cell input and output activation functions ('g' and 'h') are usually tanh or logistic sigmoid, though in some cases 'h' is the identity function. The weighted 'peephole' connections from the cell to the gates are shown with dashed lines. All other connections within the block are unweighted (or equivalently, have a fixed weight of 1.0). The only outputs from the block to the rest of the network emanate from the output gate multiplication.

Figure 32: `LSTM block` definition of [18] consisting of a single cell and its gate units.

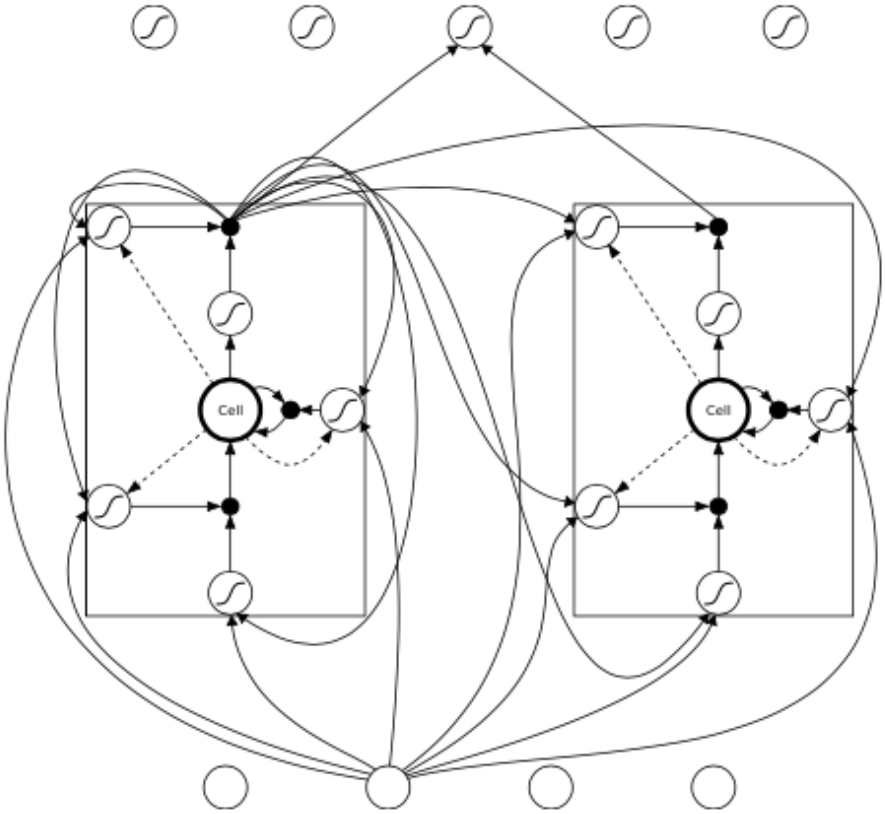


Figure 4.3: **An LSTM network.** The network consists of four input units, a hidden layer of two single-cell LSTM memory blocks and five output units. Not all connections are shown. Note that each block has four inputs but only one output.

Figure 33: `LSTM layer` consisting of multiple `LSTM blocks` following the definition of [18]

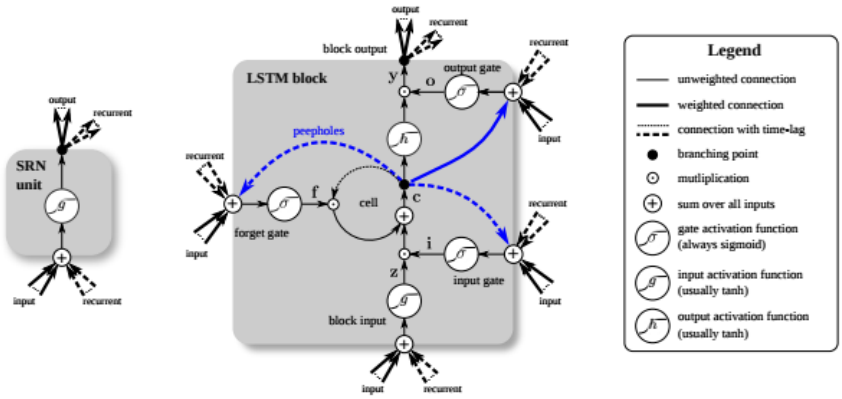


Figure 1. Detailed schematic of the Simple Recurrent Network (SRN) unit (left) and a Long Short-Term Memory block (right) as used in the hidden layers of a recurrent neural network.

Figure 34: `LSTM block` definition of [19] consisting of a single cell and its gate units.

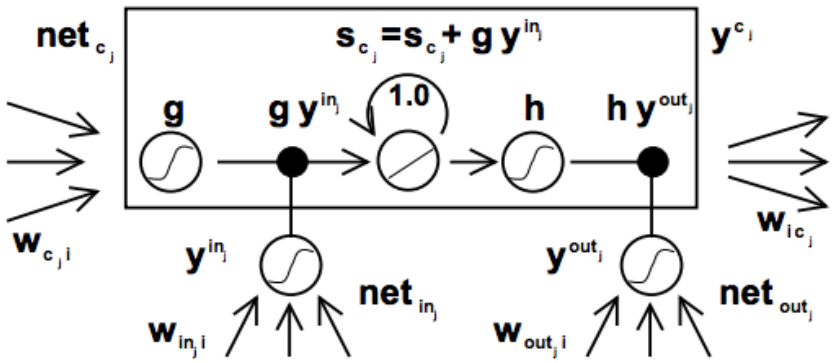


Figure 1: *Architecture of memory cell c_j (the box) and its gate units in_j, out_j. The self-recurrent connection (with weight 1.0) indicates feedback with a delay of 1 time step. It builds the basis of the "constant error carrousel" CEC. The gate units open and close access to CEC. See text and appendix A.1 for details.*

Figure 35: The `memory cell` as a self-connected, linear `unit` j in combination with an `input gate unit` and an `output gate unit` of the original LSTM paper [21]

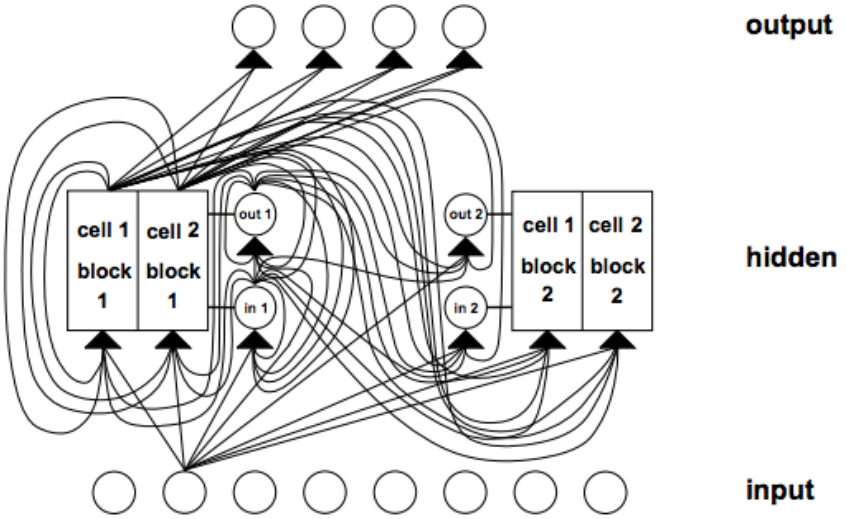


Figure 2: *Example of a net with 8 input units, 4 output units, and 2 memory cell blocks of size 2. in1 marks the input gate, out1 marks the output gate, and cell1/block1 marks the first memory cell of block 1. cell1/block1's architecture is identical to the one in Figure 1, with gate units in1 and out1 (note that by rotating Figure 1 by 90 degrees anti-clockwise, it will match with the corresponding parts of Figure 1). The example assumes dense connectivity: each gate unit and each memory cell see all non-output units. For simplicity, however, outgoing weights of only one type of unit are shown for each layer. With the efficient, truncated update rule, error flows only through connections to output units, and through fixed self-connections within cell blocks (not shown here — see Figure 1). Error flow is truncated once it "wants" to leave memory cells or gate units. Therefore, no connection shown above serves to propagate error back to the unit from which the connection originates (except for connections to output units), although the connections themselves are modifiable. That's why the truncated LSTM algorithm is so efficient, despite its ability to bridge very long time lags. See text and appendix A.1 for details. Figure 2 actually shows the architecture used for Experiment 6a — only the bias of the non-input units is omitted.*

Figure 36: `memory cell block` is comprised of multiple `memory cells` sharing input and output gates.

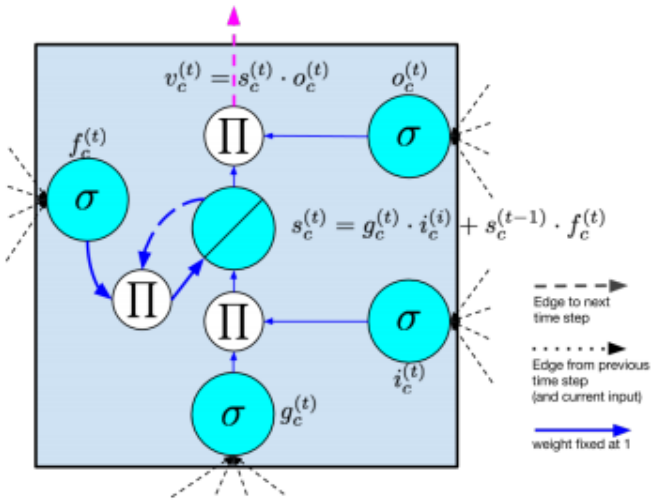


Figure 10: LSTM memory cell with a forget gate as described by Gers et al. [2000].

Figure 37: A memory cell including also the gate units following the definition of [34]

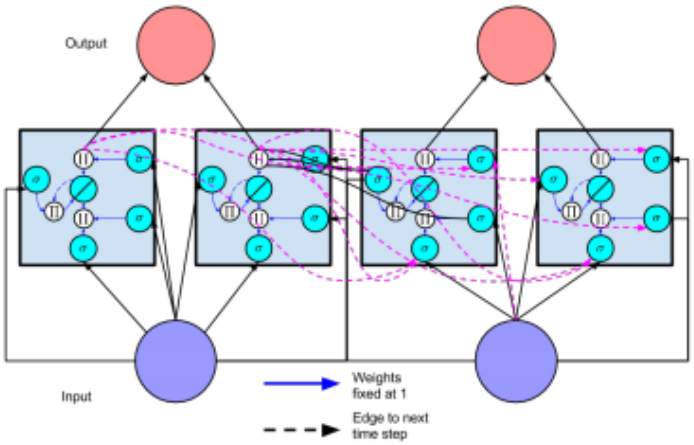


Figure 11: A recurrent neural network with a hidden layer consisting of two memory cells. The network is shown unfolded across two time steps.

Figure 38: LSTM architecture as hidden layer consisting of one or more memory cells by definition of [34]

C TensorFlow implementation

TensorFlow 1.2 addresses some issues related to building LSTM networks and is highly recommended[19].

The basic LSTM architecture of Section 4.3.2 is implemented in the `BasicLSTMCell`[20] class and the basic RNN in the `BasicRNNCell`[21]. For experiments relying on peephole connections the `LSTMCell`[22] class was used and finally, the GRU architecture is implemented in the `GRUCell`[23]

C.1 Truncating sequences and preserving state

The default TensorFlow RNN implementation will reset the states to zeros between each truncated sequence. This might not be desirable and the code below illustrates one approach to solving it:

```
def reshape_data(raw_data, n, T, d):

    data_len = len(raw_data)
    # Handle that we are doing integer division and
    # potentially missing some data by adding padding of 1
    batch_len = data_len // n + min(data_len % n, 1)

    padding = 0
    if batch_len > T:
        padding = (T - batch_len % T)

    data = np.zeros([n, batch_len + padding, d])

    for i in range(0, n):
        p = raw_data[i*batch_len:(i+1)*batch_len]
        data[i,:len(p)] = p.reshape(len(p), d)

    epoch_size = (batch_len + padding) // T

    assert((batch_len + padding) % T == 0)

    batches = np.zeros([epoch_size, n, T, d])

    for i in range(epoch_size):
        batch = data[:, i*T:(i+1)*T]
        batches[i] = batch.reshape(n, T, d)
```

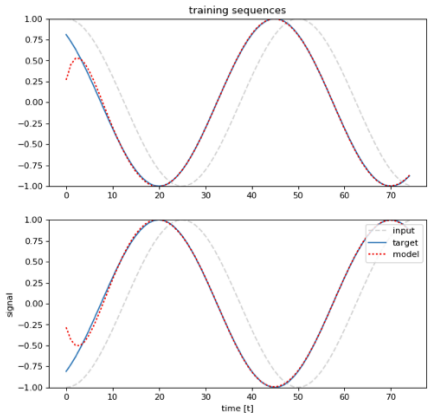
[19] https://github.com/tensorflow/tensorflow/blob/r1.4/RELEASE.mdmajor-features-and-improvements-2

[20] https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/BasicLSTMCell

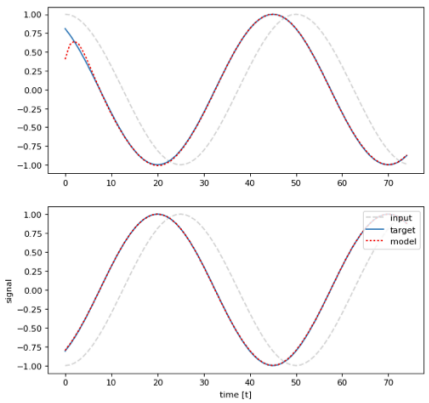
[21] https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/BasicRNNCell

[22] https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/LSTMCell

[23] https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/GRUCell



(a) Initializing states to zero. 500 epochs, shape [2, 75, 1]



(b) Passing the previous state. 500 epochs, shape [2, 1, 75, 1]

Figure 39: Burn in phase comparison between truncated sequences by passing zero state and the previous state.

```
    return batches

def run_epoch(sess, m, c, x, y):

    state = sess.run(m.initial_state)

    x = reshape_data(x, c.batch_size, c.sequence_length, c.input_size)
    y = reshape_data(y, c.batch_size, c.sequence_length, c.output_size)

    for step, batch in enumerate(x):
        _, state = sess.run([m.training_op, m.final_state],
            { m.inputs: batch, m.targets: y[step],
              m.length: x_len[step], m.initial_state: state})
```

Figure 39 depicts the solution applied to a generated sine wave time series that we should be able to learn. As can be be seen, clearly initializing states with zeros between sequences produces a burn in phase that we can eliminate be preserving the states.

C.2 Variable length time series

As well as accepting the initial state, the rnn implementation will take a parameter for length of the sequences:

```
outputs, states = tf.nn.dynamic_rnn(lstm_cell, inputs,
    sequence_length=length, initial_state=state, dtype=tf.float32)
```

C.3 Dropout

TensorFlow 1.4 address a particular problem related to drop out in LSTM networks and is highly recommended[24].

D Results

D.1 Results data sets

Upon submission of the thesis, results sets will be made available here:

- https://github.com/jensgrud/financial-forecasting-lstm

[24]https://github.com/tensorflow/tensorflow/blob/r1.4/RELEASE.mdbug-fixes-and-other-changes

D.2 Visualization

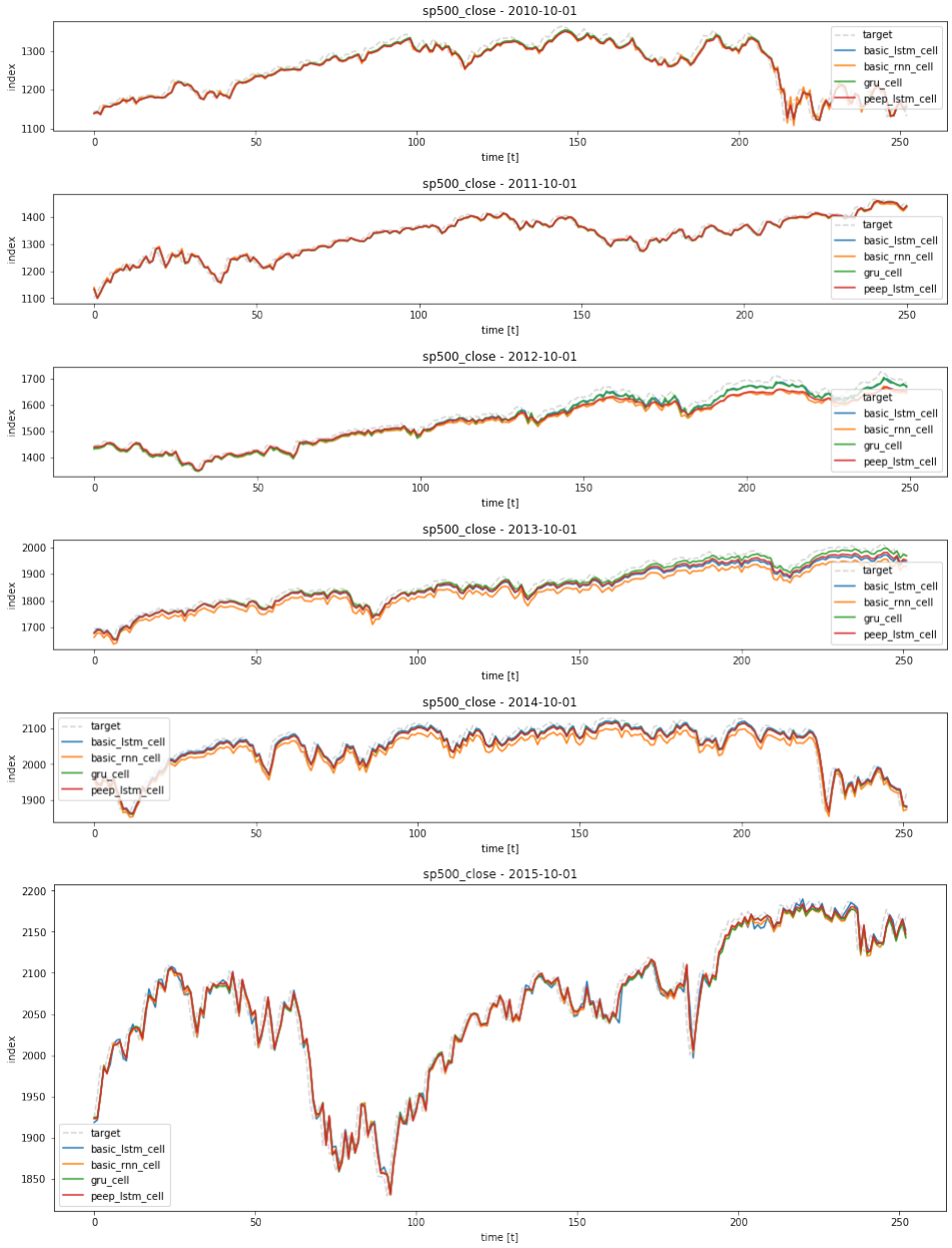


Figure 40: Target and predictions for each of the four models in the SP500 index

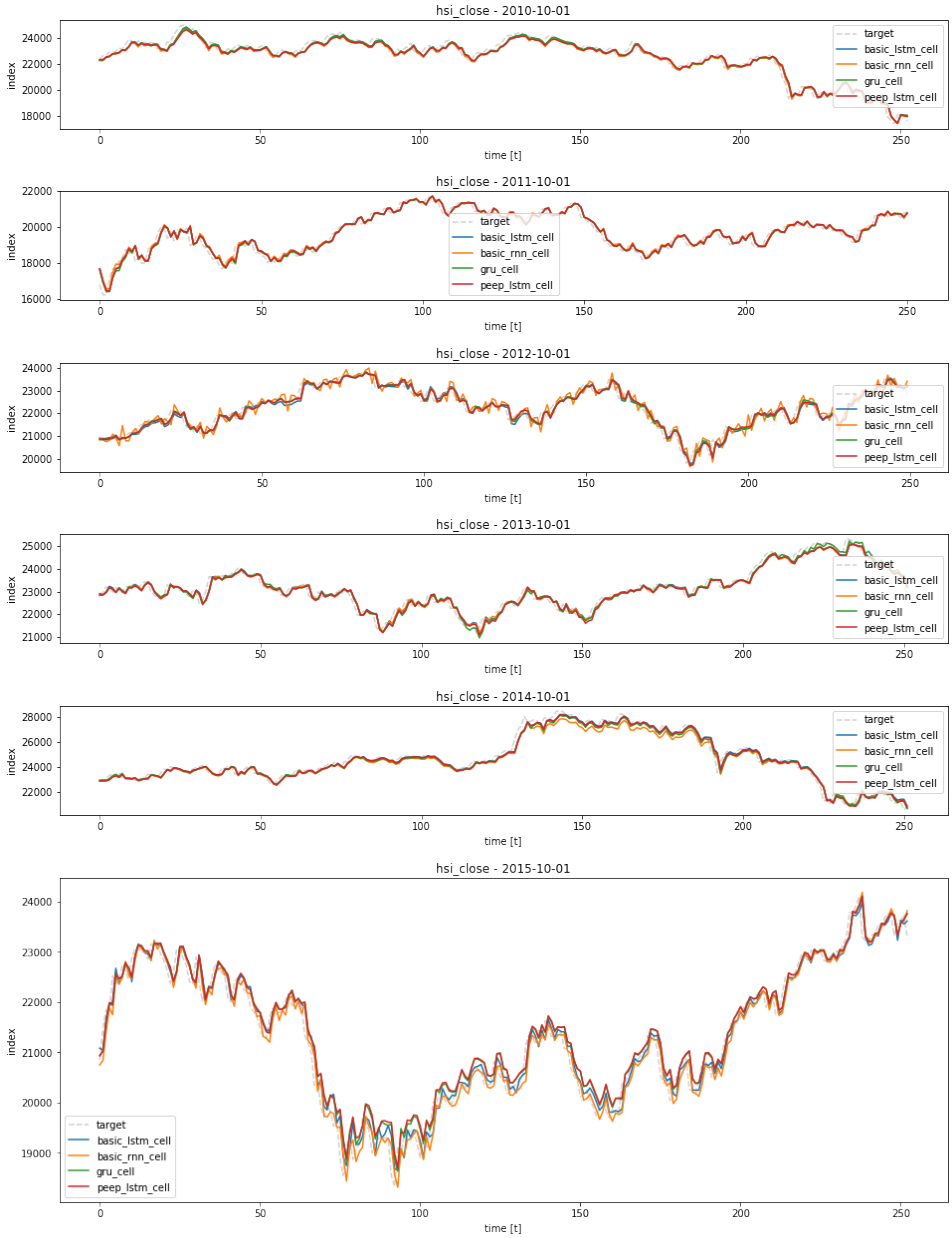


Figure 41: Target and predictions for each of the four models in the HSI index

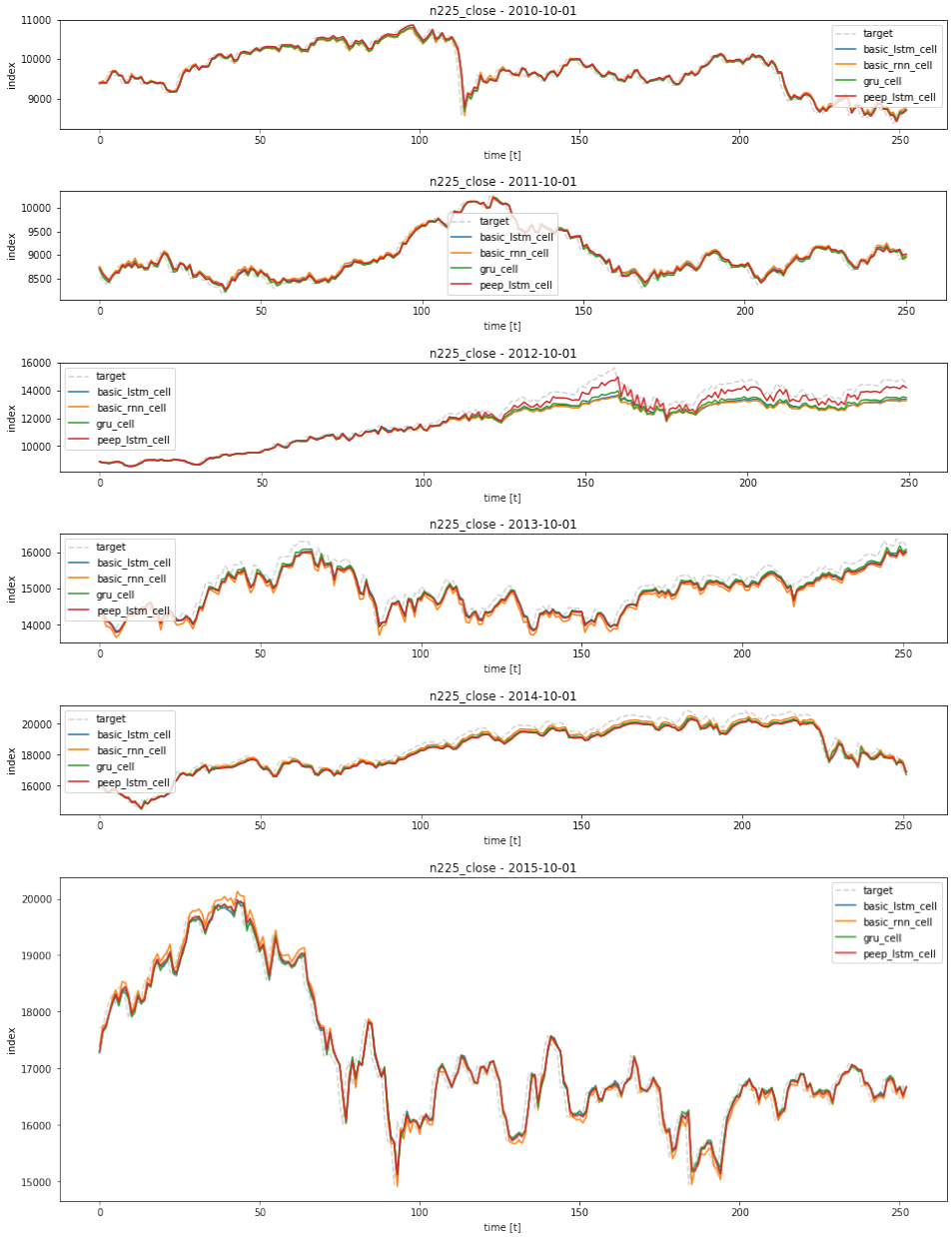


Figure 42: Target and predictions for each of the four models in the N225 index

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp,

Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Ryo Akita, Akira Yoshihara, Takashi Matsubara, and Kuniaki Uehara. Deep learning for stock prediction using numerical and textual information. In *Computer and Information Science (ICIS), 2016 IEEE/ACIS 15th International Conference on*, pages 1–6. IEEE, 2016.

[3] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016.

[4] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[5] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS ONE*, 12(7):e0180944, 2017.

[6] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[7] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of computational science*, 2(1):1–8, 2011.

[8] Kalok C Chan, G Andrew Karolyi, Francis A Longstaff, and Anthony B Sanders. An empirical comparison of alternative models of the short-term interest rate. *The journal of finance*, 47(3):1209–1227, 1992.

[9] Kai Chen, Yi Zhou, and Fangyan Dai. A lstm-based method for stock returns prediction: A case study of china stock market. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 2823–2824. IEEE, 2015.

[10] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. 2014. https://arxiv.org/pdf/1406.1078.pdf.

[11] Luca Di Persio and Oleksandr Honchar. Artificial neural networks approach to the forecast of stock market price movements. *International Journal of Economics and Management Systems*, 1:158–162, 2016.

[12] Adam Fadlalla and Chien-Hua Lin. An analysis of the applications of neural networks in finance. *Interfaces*, 31(4):112–122, 2001.

[13] Thomas Fischer and Christopher Krauß. Deep learning with long short-term memory networks for financial market predictions. Technical report, FAU Discussion Papers in Economics, 2017.

[14] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1019–1027, 2016.

[15] Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE, 2000.

[16] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[18] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer, 2012.

[19] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

[20] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. 2014.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[22] Stefano M Iacus. *Simulation and inference for stochastic differential equations: with R examples*. Springer Science & Business Media, 2009.

[23] Christian Igel and Michael Hüsken. Improving the rprop learning algorithm. In *Proceedings of the second international ICSC symposium on*

neural computation (NC 2000), volume 2000, pages 115–121. Citeseer, 2000.

[24] Christian Igel and Michael Hüsken. Empirical evaluation of the improved rprop learning algorithms. *Neurocomputing*, 50:105–123, 2003.

[25] Investotopia. Fundamental Analysis — Investotopia. https://www.investopedia.com/terms/f/fundamentalanalysis.asp, 2018. [Online; accessed 10-February-2018].

[26] Investotopia. Technical Analysis — Investotopia. https://www.investopedia.com/terms/t/technicalanalysis.asp, 2018. [Online; accessed 10-February-2018].

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015.

[28] Hengjian Jia. Investigation into the effectiveness of long short term memory networks for stock price prediction. 2016. https://arxiv.org/pdf/1603.07893.pdf.

[29] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350, 2015.

[30] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014. https://arxiv.org/pdf/1412.6980.pdf.

[31] Christopher Krauss, Xuan Anh Do, and Nicolas Huck. Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the s&p 500. *European Journal of Operational Research*, 259(2):689–702, 2017.

[32] Bjoern Krollner, Bruce Vanstone, and Gavin Finnie. Financial time series forecasting with machine learning techniques: A survey. 2010.

[33] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[34] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. 2015. https://arxiv.org/pdf/1506.00019.pdf.

[35] Nijolė Maknickienė and Algirdas Maknickas. Application of neural network for forecasting of exchange rates and forex trading. In *The 7th international scientific conference" Business and Management*, pages 10–11, 2012.

[36] Nijolė Maknickienė, Aleksandras Vytautas Rutkauskas, and Algirdas Maknickas. Investigation of financial market prediction by recurrent neural network. *Innovative Technologies for Science, Business and Education*, 2(11):3–8, 2011.

[37] Burton G Malkiel and Eugene F Fama. Efficient capital markets: A review of theory and empirical work. *The journal of Finance*, 25(2):383–417, 1970.

[38] Taesup Moon, Heeyoul Choi, Hoshik Lee, and Inchul Song. Rnndrop: A novel dropout for rnns in asr. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 65–70. IEEE, 2015.

[39] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[40] Louis B Rall. Automatic differentiation: Techniques and applications. 1981.

[41] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

[42] Riskbutler. Fintech Virtual Assistants - Machine Learning AI — Stocksneural.net. https://riskbutler.com, 2017. [Online; accessed 02-August-2017].

[43] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[44] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[45] Wikipedia. Technical analysis — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Technical%20analysisoldid=792733643, 2017. [Online; accessed 02-August-2017].

[46] Wikipedia. Coefficient of determination — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Coefficient˙of˙determination, 2018. [Online; accessed 02-February-2018].

[47] Wikipedia. Pearson correlation coefficient — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Pearson˙correlation˙coefficient, 2018. [Online; accessed 02-February-2018].

[48] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014. https://arxiv.org/pdf/1409.2329.pdf.